VICTOR 9000 TECHNICAL REFERENCE MANUAL

Victor Business Products, Inc.
June 1982

NOTICES

DISCLAIMER

Victor Business Products makes no representations or warranties of any kind with respect to the contents hereof and specifically disclaims any implied warranties or merchantability or fitness for any particular purpose. Victor Business Products shall not be liable for errors contained herein of for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Further, Victor Business Products reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Victor Business Products to notify any person of such revision or changes.

PROPRIETARY NOTICE

Publication Number 710620
Printed in U.S.A.

VICTOR® BUSINESS PRODUCTS
subsidiary of Kidde, Inc.
KIDDE

1. System Description

The VICTOR 9000 microcomputer.is designed for maximum operator comfort and ease of use. The system is composed of three modules, and occupies the desk space normally needed for an office typewriter. Its modules are: the processor unit, the display unit, and the keyboard unit. Coiled cables interconnect these standalone modules, allowing easy positioning and mobility. A standard configuration is shown in Figure 1-1.

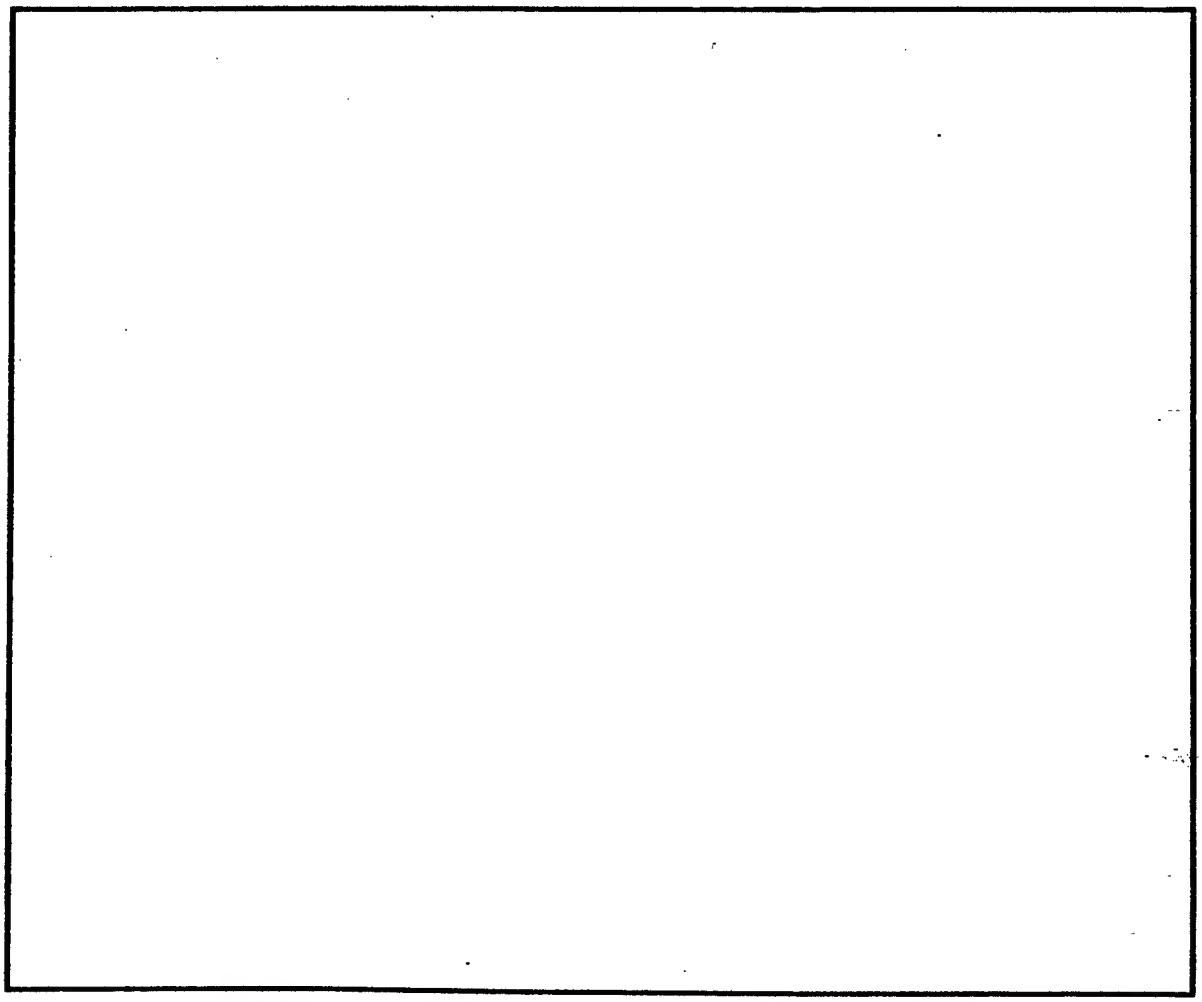

Figure 1-1. Typical Arrangement of VICTOR 9000 Components

The VICTOR 9000 can be connected to a wide variety of peripherals and accommodates local and long distance communications. Standard interfaces include a parallel port (Centronics or IEEE-488), programmable RS-232C (V-24) channels, an internal control port, and an audio controller for digitized voice and tone output.

1.1 Processor Unit

The processor unit physically supports the display unit, as shown in Figure 1-1. The main logic, disk drives, and power supply are housed in the processor unit. The two integral single-sided 5 1/4-inch floppy disk drives store up to 1.2 megabytes of information. The system incorporates a minimum 128K bytes of Random Access Memory (RAM), expandable to 512K bytes.

The heart of the VICTOR 9000 processor unit is the Intel 8088 16-bit microprocessor. This processor is a version of the Intel 16-bit 8086 processor that contains an 8-bit bus interface. The 8088 is software-compatible with the 8086, fully supporting 16-bit operations, including multiply and divide. The processor has a 20-bit physical address space, providing 1 megabyte of addressable memory.

As indicated earlier, the processor unit is the module that physically supports the display unit. It contains three basic assemblies: the main logic board, the disk drive assembly, and the power supply.

1.2 Main Logic Board

As shown in Figure 1-2, the main logic board is comprised of the central processing unit (CPU) section, the input/output (I/O) section, the display section, the disk interface section, and the Expansion bus.

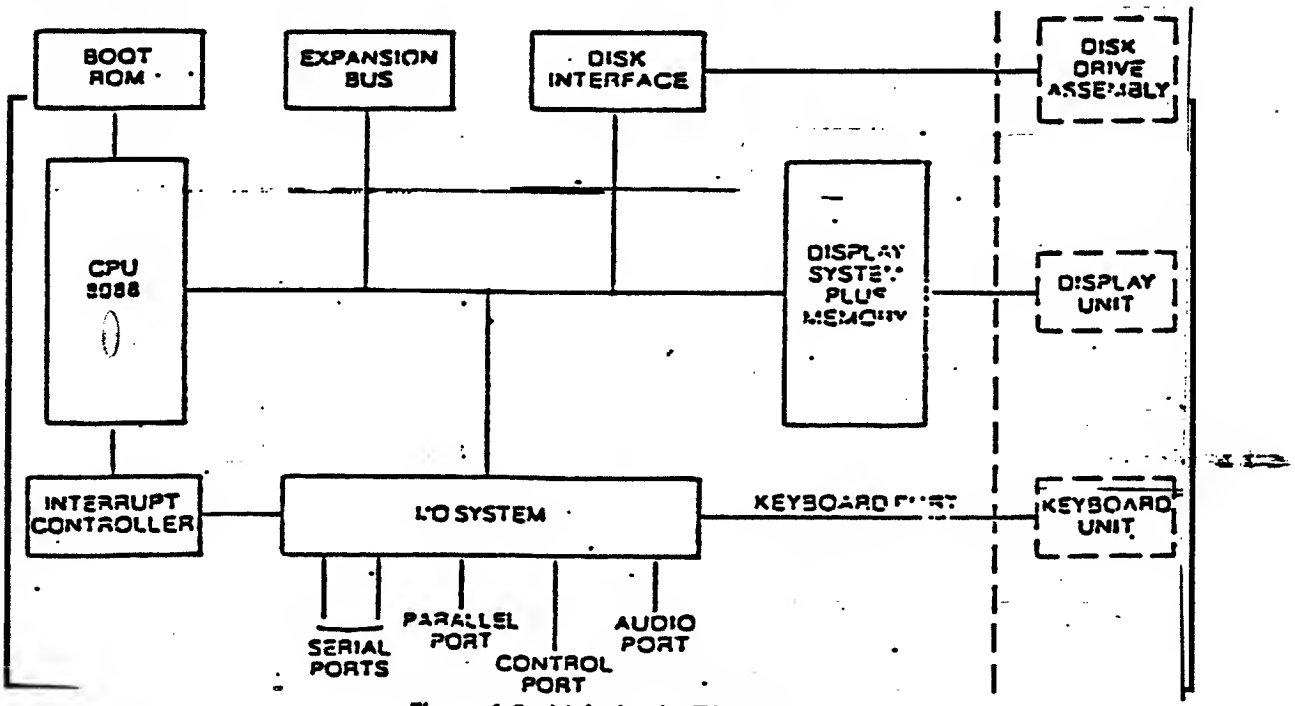

Figure 1-2. Main Logic Diagram

1.3 Display Unit

The display unit swivels and tilts to permit optimum adjustment of the viewing angle, and the unit incorporates a 12-inch antiglare screen to prevent eye strain. The display is 25 lines with each line having 80 characters.

Characters are formed in a 10×16 font cell, providing a high resolution display. A bit-mapped graphics mode with 800×400 dot matrix screen resolution is available under software control. Software also controls the overall screen brightness, character contrast, and audio volume.

The video display unit is supported by a swivel ramp and fits on top of the processor unit. The swivel ramp permits the video display unit to be swiveled right or left and to be tilted up or down. A fabric grid on the face of the CRT reduces glare and reflection and increases character contrast.

A coiled cord with a locking connector plugs the video display unit into the processor unit. The cord carries power and video signals, sync signals, and brightness control signals to the video display unit.

The video display system uses +12V power at approximately 1.2 amperes. The horizontal sweep rate is approximately 15 KHz. A vertical refresh rate of 76 Hz, or 76 frames per second, prevents visual flicker.

An interlace method of display is used. Each frame contains half the picture. This is very similar to conventional television and yields a high-resolution 400 line vertical capability.

Display brightness and contrast are both software adjustable. Brightness, controlled by signals sent from the processor unit's display section, may be varied to eight intensities. Contrast is controlled on the main logic board of the processor unit. The user may select eight levels of contrast from the keyboard.

1.4 Keyboard Unit

The keyboard unit is designed for comfort and ease of operation. It is completely software definable and features several keys that are specifically designed for special-function use in application programs. The keyboard contains separate typewriter and numeric/calculator keypad configurations, double-size general function keys, special-function keys, editing, and cursor control keys. A cluster of keys is also used to manipulate screen brightness, character contrast, and audio volume. The function of the keyboard is to generate and send coded electrical signals to the processor unit as each key is depressed or released. The keyboard is entirely reconfigurable.

The keyboard unit is approximately 19 inches wide, 1.8 inches high, and 6.4 inches deep. It is connected to the rear of the processor unit by a coiled cord.

The key switch is a high reliability capacitive-type switch on the keyboard. There is no mechanical contact. The signal is detected electrically, so the switch has a very long life.

Key surfaces are sculptured for comfortable typing. Key caps are removable and interchangeable, facilitating service and allowing the keyboard to be customized.

The keyboard unit is organized into five key groups. The central key group is arranged in a standard typewriter configuration. A numeric/calculator keypad is located at the far right of the keyboard. The general function keys across the top row are double-sized and can be defined for specific purposes by applications programs. A single column of specific function keys are located on the far left of

the keyboard. Editing and cursor control function keys are located in a double column between the typewriter keyboard and the numeric/calculator keypad groups.

The coiled cord is the conduit for all of the keyboard unit's inputs and outputs. The keyboard unit receives power and ground signals, a shield signal which protects the keyboard from static discharge and radiating noise, and three handshake or data control signals which control data transfer from the keyboard to the processor unit.

The comunication between the processor unit and the keyboard unit is serial. The transmission is in 9-bit words. The first eight bits form the data byte, with the least significant bit transmitted first. The last bit is a stop bit.

The keyboard returns key numbers and key status through the eight data bits. The most significant bit of the key number returned by the keyboard unit is status which flags a key "closed" or a key "open." The least significant seven bits are the key number.

A single-chip microprocessor in the keyboard unit scans the keyboard for key closures and communicates with the processor unit. Keyboard status communicated to the processor unit is completely independent of key condition. The microprocessor reports an event, such as a key making or breaking contact, and the processor unit determines what that key's function is, based on application program definition.

The keyboard unit processor has an event buffer. It buffers events in case activity is going on in the processor unit that prevents it from servicing all the event signals coming in.

The communication protocol is accomplished through the use of three signal lines. The first control line passes the data serially. The second control line from the keyboard indicates to the processor unit that an event signal is ready, and the processor unit acknowledges this, using the third signal as a handshake. This return line from the processor unit to the keyboard unit is called the acknowledge line. It tells the keyboard that the processor unit has taken the bit and is making the appropriate handshake.

A protocol is defined for handling overflow problems (when the keyboard unit overflows its buffer). The protocol allows the keyboard to enter a "hold-off" state, thus permitting the processor to complete an activity without losing any event signals.

The keyboard can be made to time-out and retransmit event signals in case of an error or a problem in the handshake. The keyboard processor supports N-key rollover, which means that status is reported as the keys are depressed and as they are released. As long as the event queue doesn't overflow and the processor unit keeps up with the event queue, an unlimited number of keys can be rapidly depressed.

The EU is not connected to the outside world via the system bus. It obtains instructions from a queue maintained by the BIU. When an instruction requires access to memory or to a peripheral device, the EU sends a request to the BIU to store or obtain the data. The BIU performs an address relocation that gives the EU access to a full megabyte of memory space.

2.3 Bus Interface Unit

The BIU performs all bus operations for the EU. Upon demand from the EU, the BIU transfers data between the CPU and the memory or an I/O device.

While the EU is executing instructions, the BIU fetches instructions from memory. The instructions are stored in an internal RAM array called the instruction stream queue. The 8088 instruction queue holds up to four bytes of the instruction stream. The queue size is sufficient to allow the BIU to keep the EU supplied with fetched instructions without monopolizing the system bus. The BIU fetches another instruction byte whenever:

1. one byte in the queue is empty, and
2. there is no active request for bus access (Figure 2-1).

The instruction queue usually contains at least one byte of the instruction stream; the EU does not have to wait for instructions to be fetched. The instructions in the queue are those stored in the memory locations immediately adjacent to and higher than the instruction currently being executed. That is, the queue contains the next logical instructions, as long as execution proceeds serially. If the EU executes an instruction that transfers control to another location, the BIU resets the queue, fetches the instruction from the new address, passes it immediately to the EU, and then begins refilling the queue from the new location.

The BIU suspends instruction fetching whenever the EU requests a memory or I/O read or write. A fetch already in progress is completed before the EU's bus request is executed.

2.4 General Registers

The 8088 has eight 16-bit general registers (Figure 2-3). The general registers are divided into two sets of four registers: the data registers called the H&L group (H&L stands for "high and low"), and the pointer and index registers which are called the P&I group.

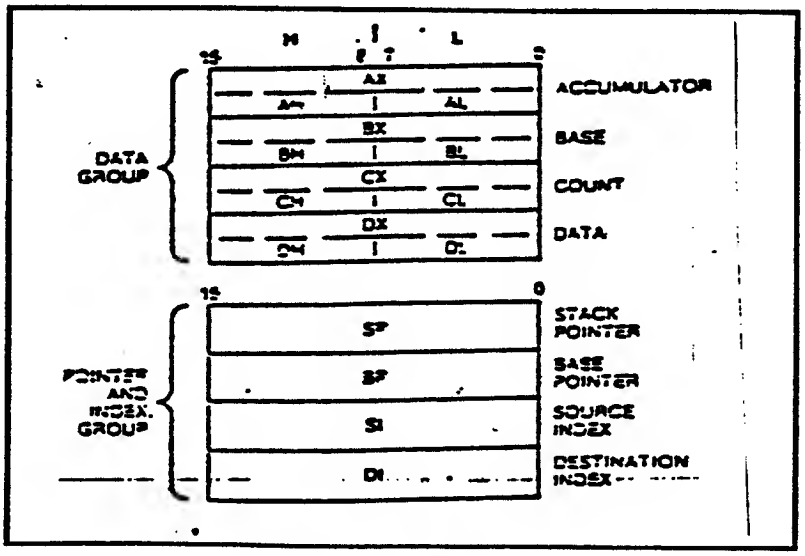

Figure 2-3. General Registers

The data registers are unique in that their upper (high) and lower halves are separately addressable. Each data register can be used interchangeably as a 16-bit register or as two 8-bit registers. However, the CPU registers are always accessed as 16-bit units. Data registers can be used without constraint in most arithmetic and logic operations. Certain instructions use specified registers implicitly (see Table 2-1), allowing compact, powerful encoding.

REGISTER	OPERATIONS
AX	Word multiply, word divide, word I/O
AL	Byte multiply, byte divide, byte I/O, translate, decimal arithmetic
AH	Byte multiply, byte divide
BX	Translate
CX	String operations, loops
CL	Variable shift and rotate
DX	Word multiply, word divide, indirect I/O
SP	Stack operations
SI	String operations
DI	String operations

Table 2-1. Implicit Use of General Registers

The pointer and index registers can also participate in most arithmetic and logic operations. All eight general registers fit the definition of "accumulator," as used with first and second generation microprocessors. The P&I registers (except for the BP register) are also used implicitly in some instructions, as shown in Table 2-1.

2.5 Segment Registers

One megabyte of memory space is divided into logical segments of up to 64K bytes each. The CPU has direct access to four segments at a time. The starting location (the base address) of each segment, is contained in the segment registers (see Figure 2-4). The CS register points to the current code segment; instructions are fetched from this segment. The SS register points to the current stack segment; stack operations are performed on locations in this segment. The DS register points to the current data segment and generally contains program variables. The ES register points to the current extra segment. (The ES register is also used for data storage.)

The segment registers can be accessed by programs and manipulated with several instructions.

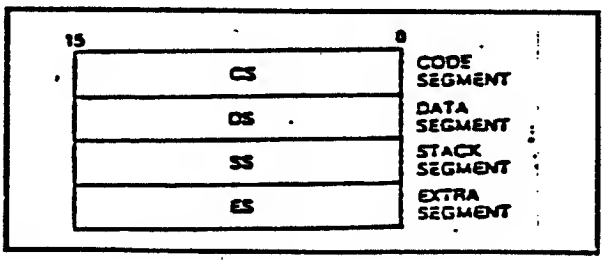

Figure 2-4. Segment Registers

2.6 Instruction Pointer

The 16-bit instruction pointer (IP) is similiar to the program counter (PC) in the 8080/8085 CPUs. The IP points to the next instruction. It is updated by the BIU so that it contains the offset (distance in bytes) of the next instruction from the beginning of the current code segment. During normal execution, the IP contains the offset of the next instruction to be fetched by the BIU. Whenever the IP is saved on the stack, it is automatically adjusted to point to the next instruction to be executed. Programs do not have direct access to the IP; however, instructions cause the IP to change and to be saved on and restored from the stack.

2.7 Flags

The 8088 has six 1-bit status flags that the EU posts (Figure 2-5). The flags reflect specified properties of the result of an arithmetic or logic operation. Different instructions affect the status flags differently. Another group of instructions is available that allows a program to alter its execution, depending on the result of a prior operation. This result is indicated by the state of these flags. Examples of conditions reflected by the flags are described below:

- The auxiliary carry flag (AF) is set when a carry out of the low nibble into the high nibble or a borrow from the high nibble into the low nibble of an 8-bit quantity (low-order byte of a 16-bit quantity) has occurred. This flag is used by decimal arithmetic instructions.

- The carry flag (CF) is set when a carry out of, or a borrow into, the high-order bit of the result (8- or 16-bit) has occurred. This flag is used by instructions that use the CF to add and subtract multibyte numbers. Rotate instructions also isolate a bit in memory or in a register by placing it in the CF.

- The overflow flag (OF) is set when an arithmetic overflow has occurred; that is, a significant digit has been lost because (i.e., the size of the result exceeded the capacity of its destination location). An interrupt on overflow instruction is available to generate an interrupt in an arithmetic overflow.

- The sign flag (SF) is set when a result's high-order bit is a 1. Negative binary numbers are represented in the 8088 in standard two's complement notation. SF indicates the sign of the result (0 = positive, 1 = negative).

- The parity flag (PF) is set when the result has even parity (an even number of 1-bits).

- The zero flag (ZF) is set when the result of the operation is 0.

Three additional control flags (Figure 2-5) can be set and cleared by programs to alter processor operations:

- Setting the direction flag (DF) causes string instructions to auto-decrement (to process strings from high addresses to low addresses). Clearing DF causes string instructions to auto-increment (to process strings from left to right).

- Setting the interrupt-enable flag (IF) allows the CPU to recognize external (maskable) interrupt requests. Clearing IF disables these interrupts. IF has no effect on nonmaskable interrupts generated externally or internally.

- Setting the trap flag (TF) puts the processor into single-step mode for debugging. In this mode, the CPU automatically generates an internal interrupt after each instruction, allowing a program to be inspected as it executes each instruction.

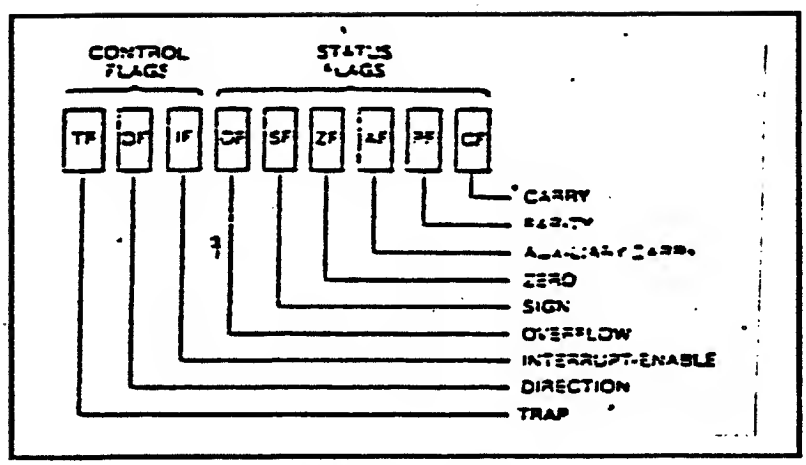

Figure 2-5. Flags

2.8 8080/8085 Register and Flag Correspondence

The registers, the flags, and the program counter in the 8080/8085 CPUs have counterparts in the 8088 CPU (see Figure 2-6). The A register (accumulator) in the 8080/8085 corresponds to the AL register in the 8088. The 8080/8085 H&L, B&C, and D&E registers correspond to registers BH, BL, CH, CL, DH, and DL, respectively, in the 8088. The 8080/8085 stack pointer (SP) and program counter (PC) correspond to the 8088 SP and IP.

The AF, CF, PF, SF, and ZF flags are the same in both CPU families. The remaining 8088 flags and registers are unique to the 8088. The 8080/8085 to 8088 mapping allows direct translation of most existing 8080/8085 program code into 8088 program code.

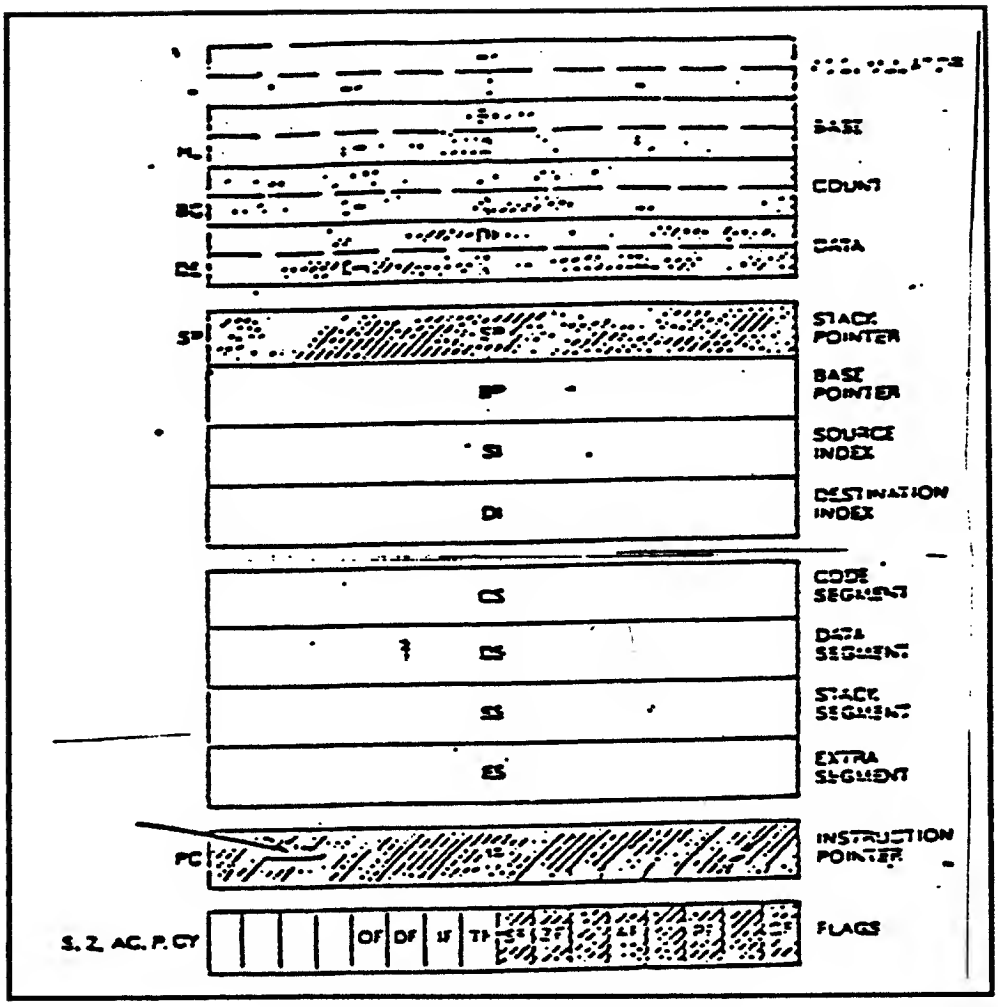

Figure 2-6. 8080/8085 Register Subset

2.9 Memory

The 8088 has 1,048,576 bytes of address space. This section describes how memory is functionally organized and used.

2.9.1 Storage Organization

The 8088 memory storage space is organized as an array of 8-bit bytes (see Figure 2-7). Instructions, byte data, and word data may be stored at any byte address, regardless of alignment. This technique saves storage space because code can be densely packed in memory (see Figure 2-8).

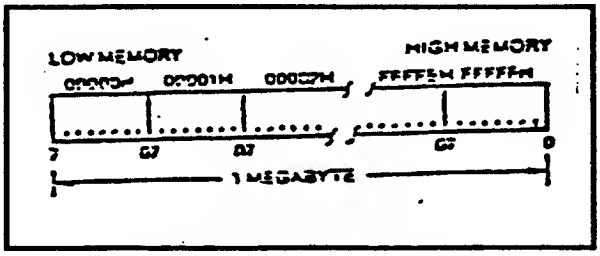

Figure 2-7. Storage Organization

The most-significant byte in word data is always stored in the higher memory location (see Figure 2-9). This storage convention is "invisible" to the user except when the user monitors the system bus or reads memory dumps. A special class of data is stored as double words (i.e., two consecutive words) called pointers, which are used to address data and code outside the currently-addressable segments. The lower-addressed word of a pointer contains an offset value, and the higher-addressed word contains a segment base address. Each word is stored conventionally with the higher-addressed byte containing the most significant eight bits of the word (see Figure 2-10).

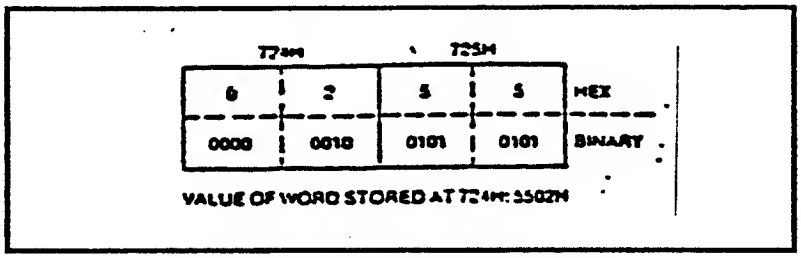

Figure 2-9. Storage of Word Variables

2.9.2 SEGMENTATION

8088 programs view the megabyte of memory space as a group of segments defined by the application. A segment is a logical unit of memory up to 64K bytes long. Each segment contains contiguous memory locations and is an independent, separately-addressable unit. Software assigns each segment a base address, which is the segment's starting location in the memory space. All segments begin on 16-byte memory boundaries. There are no other restrictions on segment locations; segments may be adjacent, disjoint, partially overlapped, or fully overlapped (see Figure 2-11). A physical memory location may be mapped into (contained in) one or more logical segments.

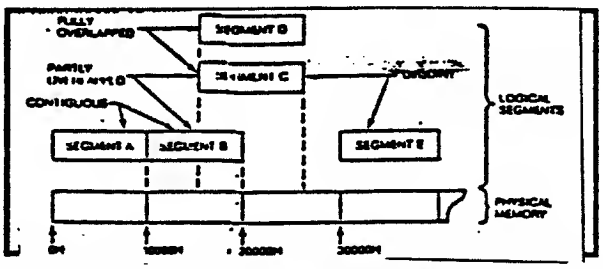

Figure 2-11. Segment Locations in Physical Memory

The segment registers contain (point to) the base address values of the four currently addressable segments (see Figure 2-12). Programs access code and data in other segments by changing the segment registers to point to the segments containing the needed code or data.

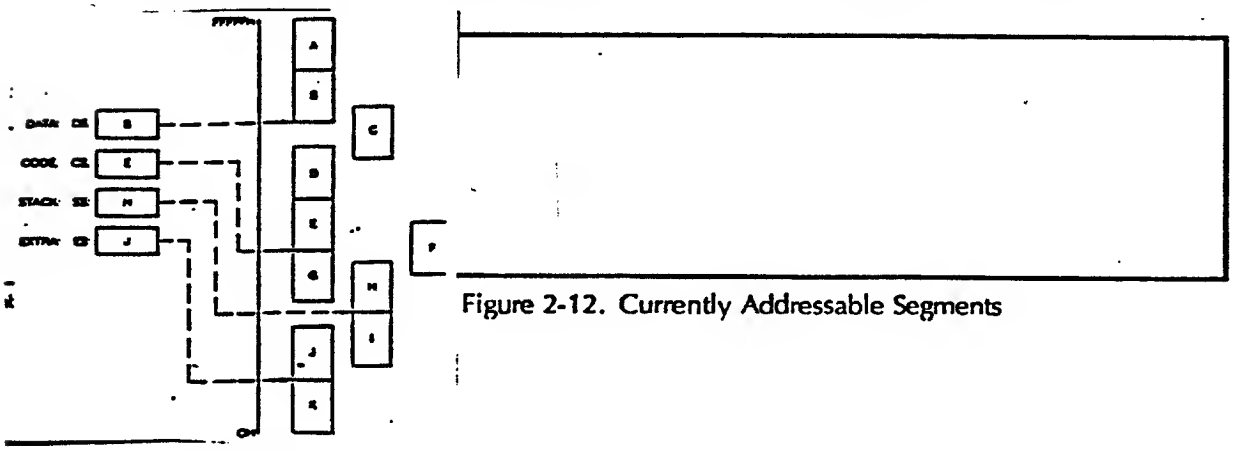

Figure 2-12. Currently Addressable Segments

Individual applications define and use segments differently. The currently-addressable segments provide a generous work space: 64KB for code, a 64KB stack, and 128KB of data storage. Many applications can be written that simply initialize the segment registers and then forget them. However, large applications should be designed with careful consideration given to segment definition.

The segmented structure of the 8088 memory space supports modular software design and discourages the development of huge, monolithic programs.

The segments can be used to advantage in many programming situations–for example, when programming an editor for several on-line terminals. A 64KB text buffer (probably an extra segment) could be assigned to each terminal. A single program could maintain all the buffers by simply changing register ES to point to the buffer of the terminal requiring service.

2.9.3 Physical Address Generation

There are two kinds of memory location addresses: physical and logical. A physical address is a 20-bit value that identifies each byte location in the megabyte memory space. The physical address range varies from 0 through $FFFFF_{16}$. All exchanges between the CPU and memory components use physical addresses.

Programs use logical addresses, which allow code to be developed before the code is assigned physical addresses. This technique facilitates dynamic management of memory resources.

A logical address consists of two values: a segment-base value and an offset value. The segment-base value for any memory location is the value that defines the first byte of the segment. The offset value is the number of bytes from the beginning of the segment to the target location. Segment-base and offset values are unsigned 16-bit quantities. The lowest addressed byte in a segment has an offset value of 0. Different logical addresses can map to the same physical location, as shown in Figure 2-13. The physical memory location $2C3_{16}$ shown in Figure 2-13 is contained in two different overlapping segments, one beginning at $2B0_{16}$ and the other at $2C0_{16}$.

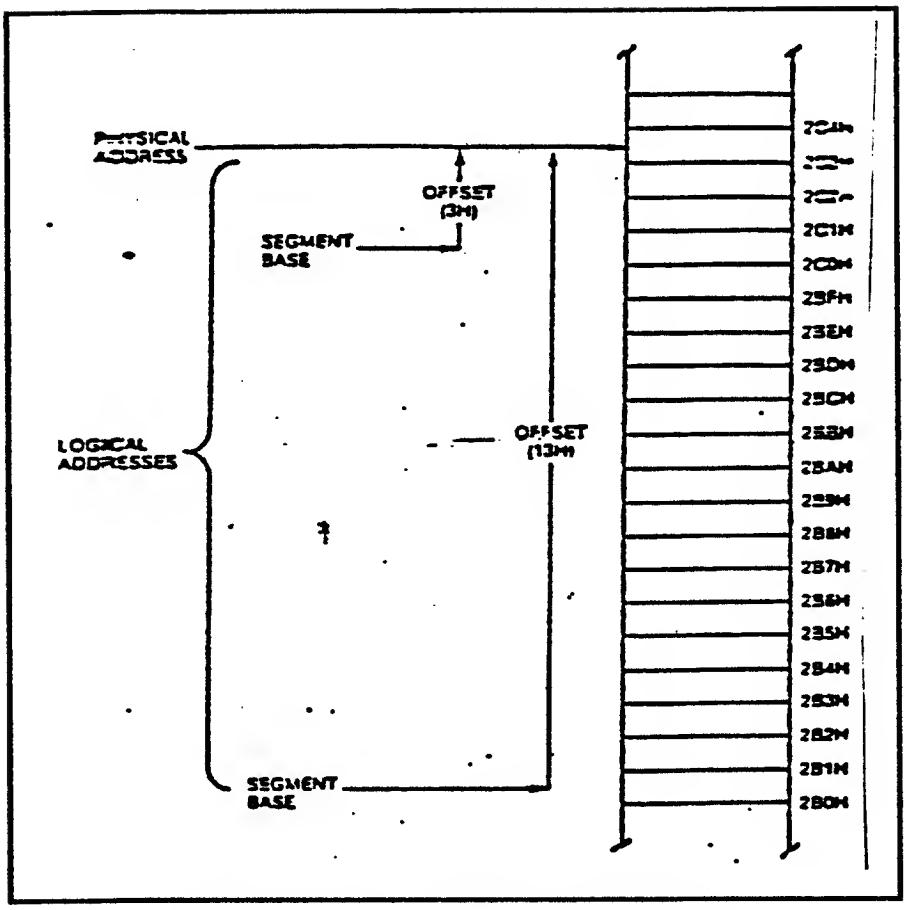

Figure 2-13. Logical and Physical Addresses

When the BIU accesses memory to fetch an instruction, or to obtain or store a variable, it generates a physical address from a logical address. It does this by (1) shifting the segment-base value four bit positions, and (2) adding the offset value, as illustrated in Figure 2-14. This addition process results in modulo 64K addressing, which causes addresses to wrap around from the end of a segment to the beginning of the same segment.

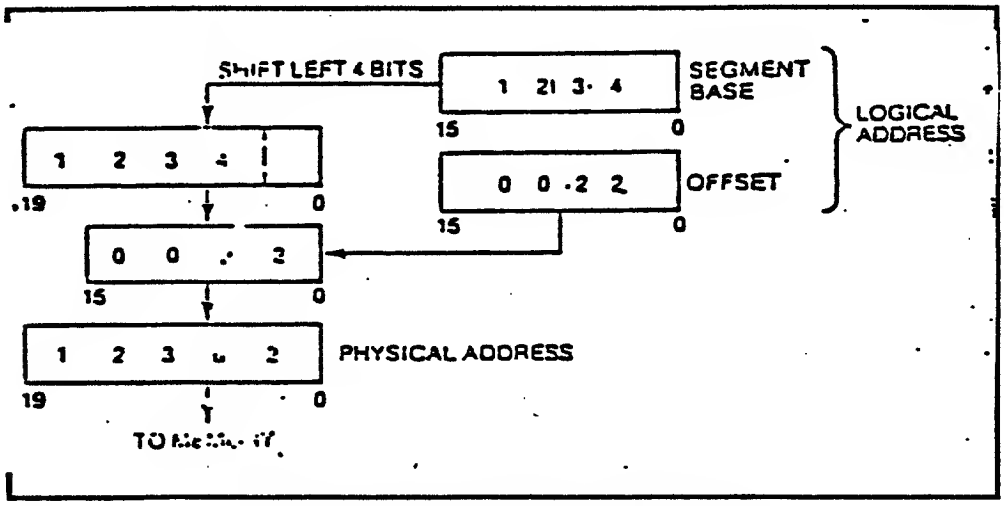

Figure 2-14. Physical Address Generation

The BIU obtains the logical address of a memory location from different sources, depending on the type of reference that is being made (see Table 2-2). Instructions are always fetched from the current code segment. The IP contains the offset of the target instruction from the beginning of the segment. Stack instructions always operate on the current stack segment. The SP contains the offset of the top of the stack. Most memory operands reside in the current data segment, although the program can instruct the BIU to access a variable in one of the other currently addressable segments. The offset of a memory variable is calculated by the EU; the calculation is based on the addressing mode specified in the instruction, and the result is called the operand's effective address (EA).

TYPE OF MEMORY REFERENCE	BASE	DEFAULT SEGMENT BASE	ALTERNATE SEGMENT OFFSET
Instruction fetch	CS	NONE	IP
Stack operation	SS	NONE	SP
Variable (except following)	DS	CS,ES,SS	Effective address
String source	DS	CS,ES,SS	SI
String destination	ES	NONE	DI
BP used as base register	SS	CS,DS,ES	Effective Address

Table 2-2. Logical Address Sources

Strings are addressed differently than other variables. The source operand of a string instruction usualiy lies in the current data segment; however, another currently-addressable data segment may be specified. The source operand's offset is taken from register SI (the source index register). The destination operand of a string instruction always resides in the current extra segment, and its offset is taken from DI (the destination index register). The string instructions automatically adjust SI and DI as they process the strings one byte or word at a time.

When register BP (the base pointer register) is designated as a base register in an instruction, the variable is assumed to reside in the current stack segment. Using register BP is a convenient way to address data on the stack. The BP register can be used to access data in any of the other currently addressable segments.

Programmers usually find the segment assumptions of the BIU convenient to use. A programmer can, however, direct the BIU to access a variable in any of the currently-addressable segments by preceding an instruction with a segment override prefix. This 1-byte machine instruction tells the BIU which segment register to use to access a variable referenced in the following instructions. The only exception to this is a string instruction's destination operand, which must be located in the extra segment.

2.9.4 Dynamically Relocatable Code

Dynamically relocatable–or position-independent–programming is made possible by the segmented memory structure of the 8088. The dynamic relocation technique makes effective use of available memory by taking advantage of the system's multiprogramming/multitasking capabilities. Inactive programs can be written to disk, making the space they occupied available to other programs. A disk-resident program can be read back into any available memory location and restarted.

When a program needs a large contiguous block of storage and only nonadjacent fragments are available, other program segments can be compacted to free up a contiguous space (Figure 2-15).

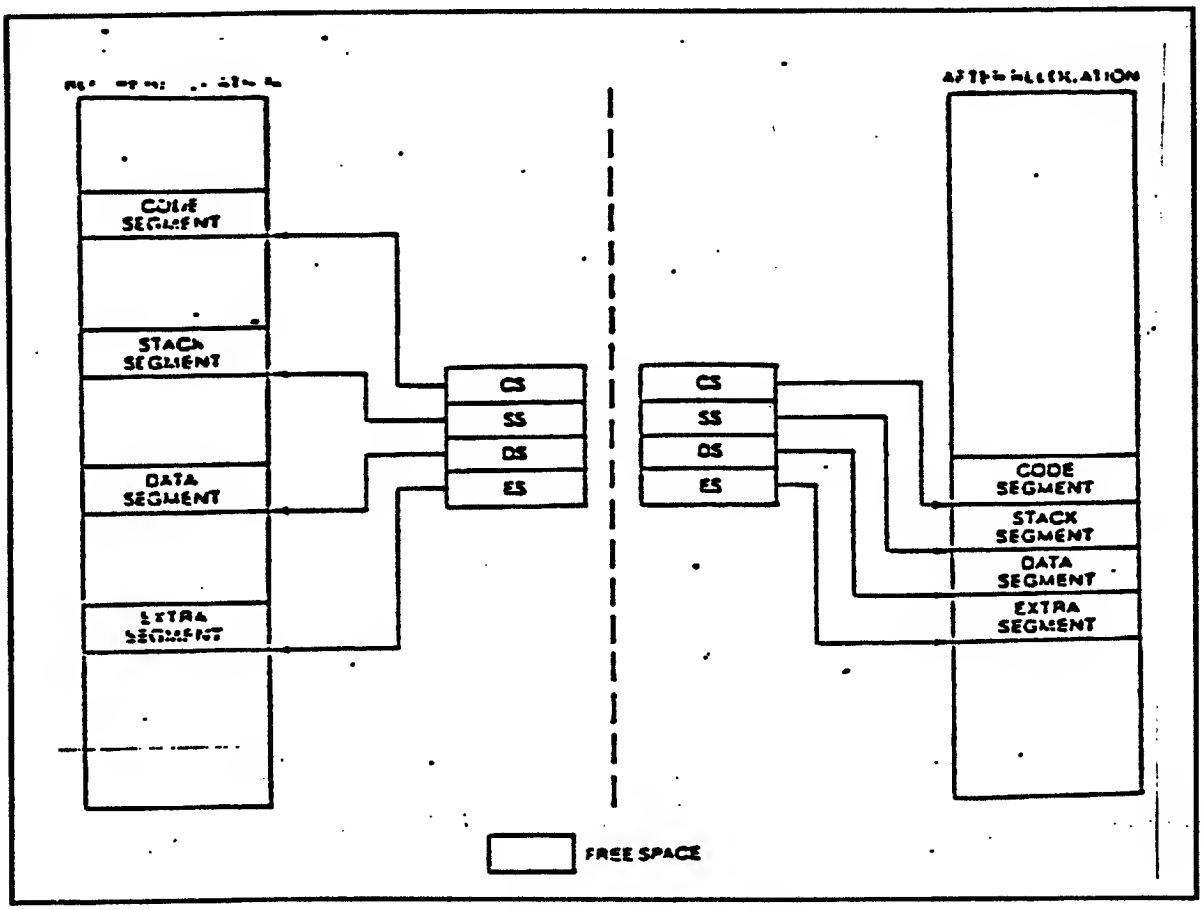

Figure 2-15. Dynamic Code Relocation

To be dynamically relocatable, all offsets in the program must be relative to fixed values contained in the segment registers. This allows the program to be moved anywhere in memory as long as the segment registers are updated to point to the new base addresses. A dynamically relocatable program must not load o alter its segment registers and must not transfer directly to a location outside the current code segment.

2.9.5 Stack Implementation

Stacks in the 8088 are implemented in memory. They are located by the SS (the stack segment register) and the SP (the stack pointer register). A system may have an unlimited number of stacks. Each may be the maximum length of a segment, 64K bytes. Attempting to expand a stack beyond 64K bytes overwrites the beginning of the stack. Only one stack is directly addressable at a time; this stack is the current stack, often referred to simply as "the" stack. SS contains the base address of the current stack. SP contains the offset of the top of the stack from the stack segment's base address. The stack's base address (contained in SS) is not the "bottom" of the stack.

Stacks are 16 bits wide. Instructions that operate on a stack add and remove stack items one word at a time. An item is pushed onto the stack (see Figure 2-16) by decrementing SP by 2 and writing the item at the new TOS (top of stack). An item is popped off the stack by copying it from TOS and then incrementing SP by 2. In other words, the stack grows down in memory toward its base address. Stack operations never move or erase items on the stack. The TOS changes only as a result of updating the stack pointer.

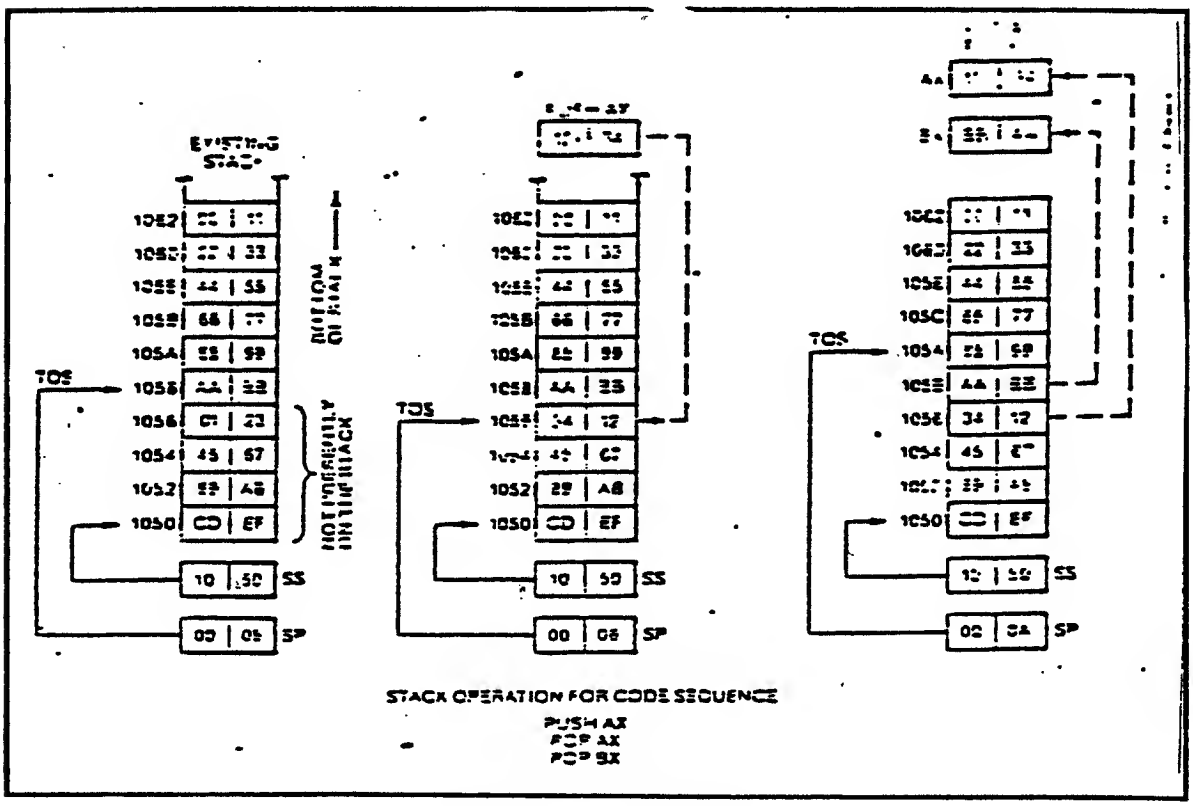

Figure 2-16. Stack Operation

2.9.6 Dedicated and Reserved Memory Locations

Two areas in extremely low and high memory—0 through $7F_{16}$ (128 bytes) and $FFFFO_{16}$ through $FFFFF_{16}$ (16 bytes)—are dedicated to specific processor functions or are reserved for use by hardware and software products (Figure 2-17). These areas are reserved for interrupt and system reset processing, and should not be used for any other purpose.

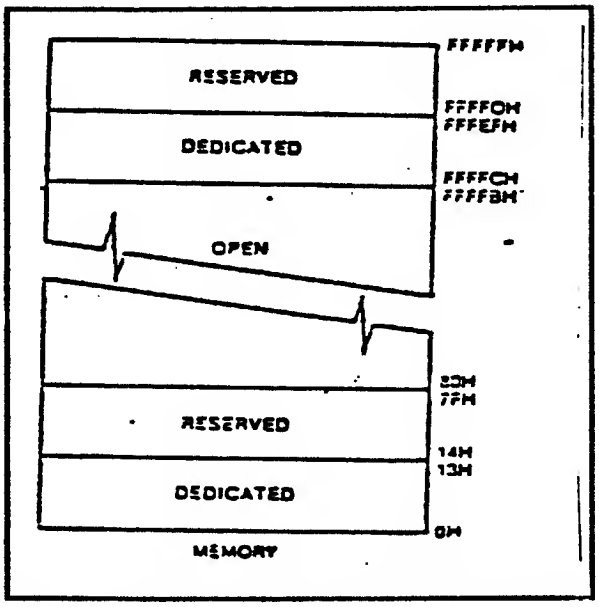

Figure 2-17. Reserved and Dedicated Memory Locations

2.9.7 8086/8088 Memory Access

The 8088 always accesses memory in bytes. Word operands are accessed in two bus cycles, regardless of their alignment. Instructions are also fetched one byte at a time. Although word-operand alignment does not affect performance, locating 16-bit data on even addresses ensures maximum throughput if the system is transferred to an 8086.

3. Communications Controller

3.1 Introduction

Serial communication in the VICTOR 9000 is handled by a complex semiconductor device, the NEC μPD7201 Multiprotocol Serial Communications Controller (MPSC²) built by Nippon Electric Corporation. This is a versatile device designed to give you high-level control of your data communication protocols with maximum flexibility and minimum processor overhead. The MPSC² contains two complete full duplex channels in a 40 pin package and incorporates a variety of sophisticated features to simplify your protocol management. The chapter describes in detail the way the device is used in the VICTOR 9000.

3.2 Features

Implements the three vasic data/communications protocols
- Asynchronous
- Character-oriented synchronous (monosync, bisync, external sync)
- Bit-oriented synchronous (SDLC/HDLC)

Provides extensive error checking
- Parity
- CRC-16
- CRC-CCITT
- Break/Abort detection
- Framing Error detection

Enhanced data reliability
- Double-buffered transmitters
- Quadrupally-buffered receivers
- Programmable transmitter underrun handling

3.3 Signal Descriptions

This section describes the various signal functions available on the MPSC².

D0-D7	The Data Bus lines are connected to the system data bus. Data or status from the MPSC² is output on these lines when $\overline{CS}$ and $\overline{RD}$ are active and data or commands are latched into the MPSC² on the rising edge of $\overline{WR}$ when $\overline{CS}$ is active.
CS	Chip Select allows the MPSC² to transfer data or commands during a read or write cycle.
B/$\overline{A}$	Channel Select - A low selects channel A. A high selects channel B for access during a read or write cycle.
C/$\overline{D}$	Control/Data Select with $\overline{RD}$, $\overline{WR}$ and B/$\overline{A}$, selects the data registers (C/$\overline{D}$ ≤ 0) or the Control and Status Registers (C/$\overline{D}$ eq 1) for access over the data bus.
$\overline{RD}$	Read (with either $\overline{CS}$ during a read cycle or $\overline{HAI}$ during a DMA cycle) notifies the MPSC² to read data or status from the device.
$\overline{WR}$	Write (with either $\overline{CS}$ during a read cycle or $\overline{HAI}$ during a DMA cycle) notifies the MPSC² to write data or control information to the device.
$\overline{RESET}$	A low on this input (one complete CLK cycle minimum) initializes the MPSC² to the following conditions: receivers and transmitters disabled, TxDA and TxDB set to marking (high), and Modem Control Outputs $\overline{DTRA}$, $\overline{DTRB}$, $\overline{RTSA}$, $\overline{RTSB}$ set high. Additionally, all interrupts are disabled, and all interrupt and DMA Requests are cleared. After $\overline{RESET}$, you must rewrite all Control Registers before restarting operation.
CLK	A TTL-level system clock signal is applied to this input. The system clock frequency must be at least 4.5 times the data clock frequency applied to any of the Data Clock inputs $\overline{TxCA}$, $\overline{TxCB}$, $\overline{RxCA}$ or $\overline{RxCB}$.
$\overline{INT}$	Interrupt Request $\overline{INT}$ is pulled low when an internal interrupt request is accepted.
$\overline{INTA}$	Interrupt Acknowledge. The processor gererates two or three $\overline{INTA}$ pulses (depending on the processor type) to signal all peripheral devices that an Interrupt Acknowledge Sequence is taking place. During the interrupt acknowledge sequence the MPSC², if so programmed, places information on the data bus to vector the processor to the appropriate interrupt service location.
$\overline{PRI}$	Interrupt Priority In informs the MPSC² whether the gighest priotity device is requesting interrupt and is used with $\overline{PRO}$ to implement a priority resolution "daisy chain" when there is mor than one interrupting device. The state of $\overline{PRI}$ and the programmed interrupt mode determine the MPSC²s response to and Interrupt Acknowledge Sequence.
$\overline{PRO}$	Interrupt Priority Out is active when $\overline{HAI}$ is active and the MPSC² is not requesting interrupt ($\overline{INT}$ is inactive). The active state informs the next lower

priority device that there are no higher priority interrupt requests pending during an Interrupt Acknowledge Sequence.

$\overline{WAITA}$,$\overline{WAITB}$ — These outputs synchronize the processor with the MPSC² when block transfer mode is used. You may program it to operate with either the Receiver or Transmitter, but not both simultaneously. $\overline{WAIT}$ is normally inactive. For example, if the processor tries to perform an inappripriate data transfer such as a write to the Transmitter when the Transmitter Buffer is full, the $\overline{WAIT}$ output tor that channel is active until the MPSC² is ready to accept the data. The $\overline{CS}$, C/$\overline{D}$, B/$\overline{A}$, $\overline{RD}$ and $\overline{WR}$ inputs must remain stable while $\overline{WAIT}$ is active.

DRQTxA, DRQRxA, DRQRxB — When DMA Request lines are active, they indicate to a DMA Controller that a Transmitter or Receiver is requesting a DMA data transfer.

$\overline{HAI}$ — Hold Acknowledge In notifies the MPSC² that the host processor has acknowledged the DMA request and has placed itself in the hold state. The MPSC² then performs a DMA cycle for the highest priority outstanding DMA request, if any.

$\overline{HAO}$ — Hold Acknowledge Out with $\overline{HAI}$, implements a priority daisy chain for multiple DMA devices. $\overline{HAO}$ is active when $\overline{HAI}$ is active and there are no DMA requests pending in the MPSC².

$\overline{TxDA}$,$\overline{TxDB}$ — Transmit Data. Serial data from the MPSC² is output on these pins.

$\overline{TxCA}$,$\overline{txCB}$ — The Transmitter Clocks control the rate at which data is shifted out at TxD. You may program the MPSC² so that the Clock rate is 1×, 16×, 32×, or 64× the data rate. Data changes on the falling edge of $\overline{TxC}$. TxC features a Schmitt-trigger input for relaxed rise and fall time requirements.

RxDA,RxDB — Receiver Data. Serial data to the MPSC² is input on these pins.

$\overline{RxCA}$,$\overline{RxCB}$ — Receiver Clocks control the sampling and shifting of serial data at RxD. You may program the MPSC² so that the clock rate is 1×, 16×, 32×, or 64× the data rate. RxD is sampled on the rising edge of $\overline{RxC}$. $\overline{RxC}$ features a Schmitt-trigger input for relaxed rise and fall time requirements.

$\overline{DTRA}$,$\overline{DTRB}$ — Data Terminal Ready pins are general-purpose outputs which may be set or reset with commands to the MPSC².

$\overline{RTSA}$,$\overline{RTSB}$ — Request to Send. When you operate the MPSC² in one of the Synchronous modes, $\overline{RTSA}$ and $\overline{RTSB}$ are general-purpose outputs that you may set or reset with commands to the MPSC². In Asynchronous mode, $\overline{RTS}$ is active immediately as soom as it is programmed on. However, when programmed off, $\overline{RTS}$ remains active until the Transmitter is completely empty. This feature simplifies the programming required to perform modem control.

$\overline{DCDA}$,$\overline{DCDB}$ — Data Carrier Detect generally indicates the presence of valid serial data at RxD. You may program the MPSC² so that the Receiver is enabled only when $\overline{DCD}$ is low. You may also program the MPSC² so that any change in state that lasts longer than the minimum specified pulse width causes an interrupt and latches the $\overline{DCD}$ status bit to the new state.

$\overline{CTSA}$,$\overline{CTSB}$ — Clear to Send generally indicates that the receiving modem or peripheral is ready to receive data from the MPSC². You may program the MPSC² so that the transmitter is enabled only when $\overline{CTS}$ is low. As with $\overline{DCD}$, you may program the MPSC² to cause an interrupt and latch the new state when $\overline{CTS}$ changes state for longer than the minimum specified pulse width.

SYNCA,SYNCB — The function of the $\overline{SYNC}$ pin depends upon the MPSC² operating mode. In asynchronous mode, SYNC is an input that the processor can read. It can be programmed to generate an interrupt in the same manner as $\overline{DCD}$ and $\overline{CTS}$.

In External sync mode, SYNC is an input which notifies the MPSC² that synchronization has been achieved (see Figure 3-1 for detailed timing). Once synchronization is achieved, hold SYNC low until synchronization is lost or a new message is about to start.

In Internal Synchronization modes (monosync, bisync, SDLC), SYNC is an output which is active wherever a SYNC character match is made (see Figure 3-2 for detailed timing). There is no qualifying logic associated with this function. Regardless of character boundaries, SYNC is active on any match.

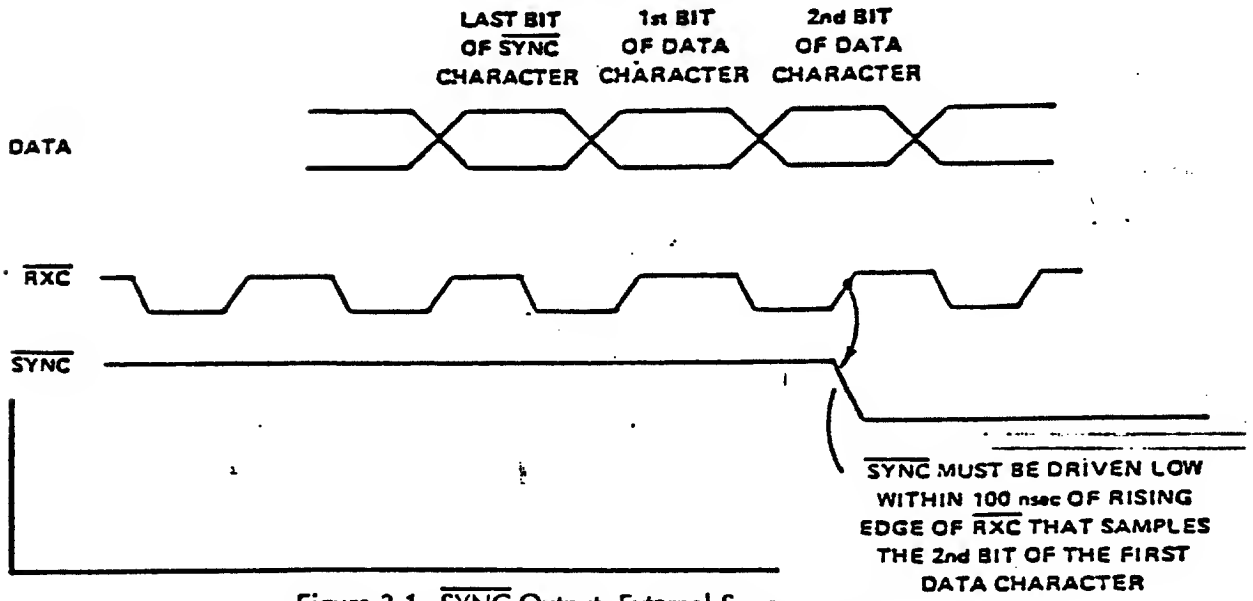

Figure 3-1. $\overline{SYNC}$ Output, External Sync...

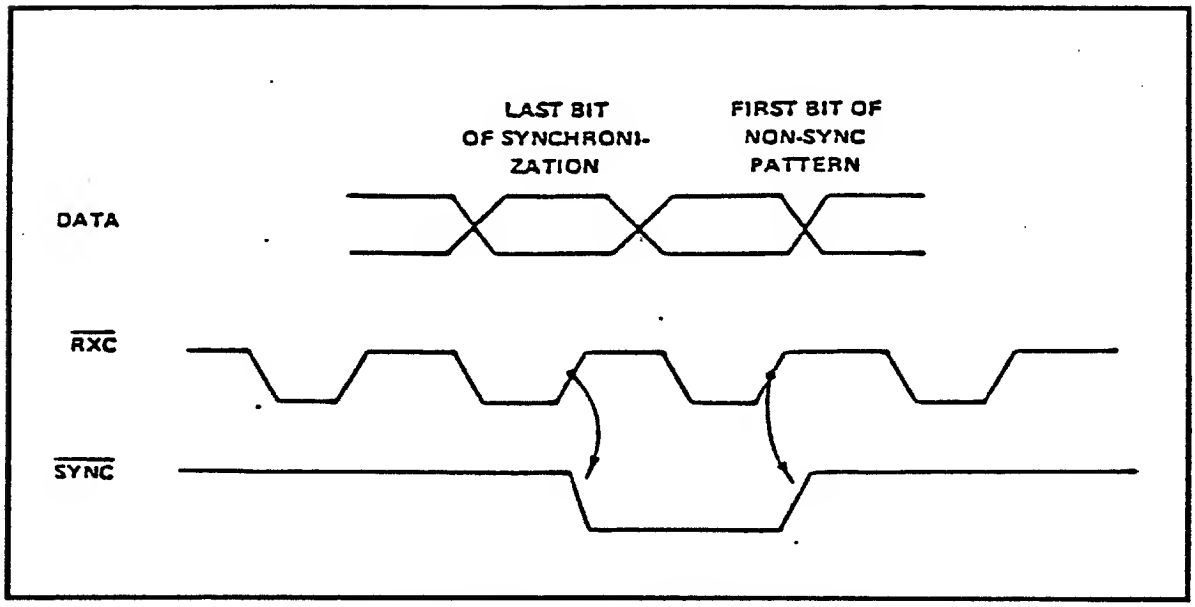

Figure 3-2. SYNC Output, Internal Synchronization

3.4 Protocols

A protocol defines a set of rules for transmitting information and control from one place to another. In parallel protocols as you might find on a microprocessor bus, dedicated "control" lines handle functions such as timing, type of information, and error checking. Since the object of werial data communications is to minimize the number of wires, the protocol used must place all of this control information in the serial data stream.

The basic protocol unit or frame can be built into increasingly complex protocols by definig special control characters and fields, and by grouping frames together into larger units. Virtually all communications protocols currently in use are based on one of three basic protocols: Asynchronous, Synchronous, Character or Count-Oriented Protocols (COPs), and Bit-Oriented Protocols (BOPs).

3.4.1 Asychronous Protocol

In asychronous protocol, each character transmitted has its own framing information in the form of a Start and Stop Bit(s). Each character is a "message" in itself and may be asynchronous with respect to any others. You can implement error detection by adding a parity bit to each character. The transmitter makes the parity bit 1 or 0 so that the character plus parity contains an even or odd number of ones for Even Parity or Odd Parity, respectively. Figure 3-3 illustrates the asynchronous data format.

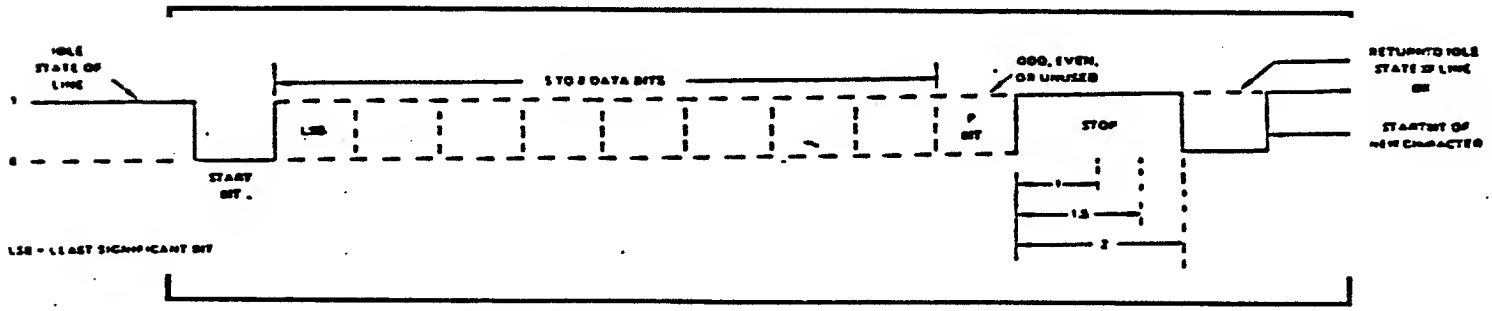

Figure 3-3 Asynchronous Data Character Format

3.4.2 Synchronous Character Oriented Protocols

In Synchronous Character Oriented Protocols (COPs), the start and stop bits associated with each character are eliminated. A synchronization (sync) character that is not part of the data is transmitted before the data to establish proper framing. The synchronization character is usually 8 or 16 bits long. Monosync and IBM BISYNC are typical examples of COP's (Figure 3-4). Since the framing information is presented only at the beginning, the transmitter must insert fill characters to maintain synchronization. Sync characters are commonly used for this purpose.

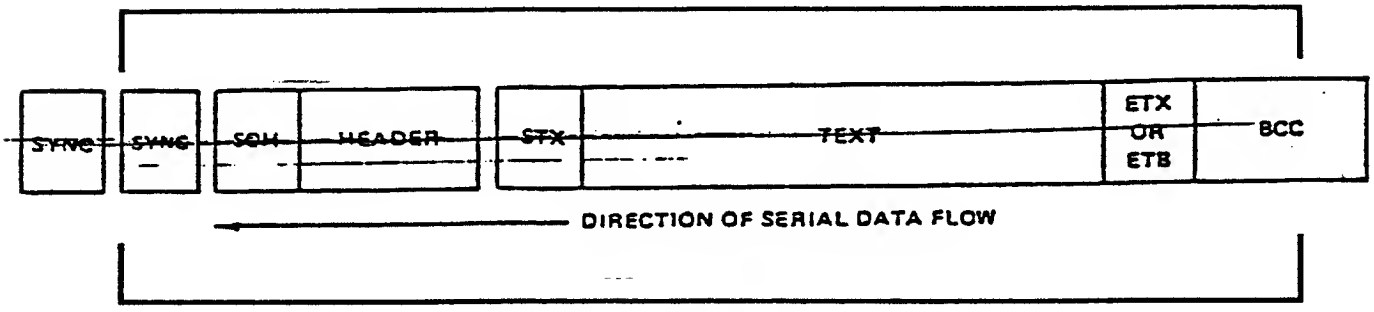

Figure 3-4 BISYNC Message Format

As with the asynchronous protocol, a parity bit may be used with each character to provide error checking. A more reliable check is performed by calculating a special 16-bit Block Check Character called a Cyclic Redundancy Check (CRC) for the entire data block and transmitting this character at the end of the data. The nost commonly used CRC polynomial for COP's is called CRC-16.

A disadvantage of the character-oriented protocol is having to use special characters such as SYNC to define various portions of a message when you send non-character binary data ("transparent data" in bisync terminology). to do this, you must transmit special DLE sequences and selectively exclude certain characters from the CRC calcula;tion for both the transmitter and receiver. The MPSC^2 features special dircuitry to simplify this operation.

3.4.3 Synchronous Bit-Oriented Protocols

Synchronous Bit-Oriented Protocols (BOP's) use a special set of rules to distinguish between data and framing characters. This eliminates some of the problems associated with COP's. The most common BOP's in use are the almost-identical HDLC and SDLC protocols shown in Figure 15.5.

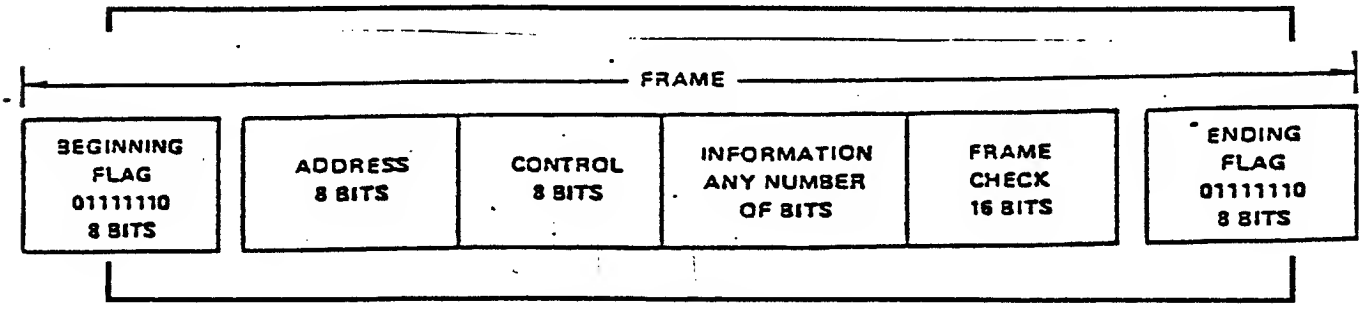

Figure 3-5 Basic SDLC Frame

The rules for SDLC (henceforth we will refer only to SDLC although the same information applies to HDLC as well) are quite simple. The basic transmission unit is called a "frame" and is deliniated by a special flag character 01111110 (flags cannot be used as filler like the COP sync character). The data or information field may consist of any number of bits; not necessarily an integral number of n-bit characters. Since data could contain the 01111110 pattern, the tranxmitter performs the following operation: if five consecutive ones are transmitted, the transmitter inserts a zero bit before the next data bit. Likewise, the receiver must delete any zero that follows five consecutive ones. Six consecutive ones indicate a flag character and eight or more ones indicate a special abort condition.

Error checking is done with a 16-bit CRC character inserted between the end of the information field and the End Of Frame flag. The CRC-CCITT polynomial is generally used. The end of a frame is determined by counting 16 bits (CRC) back from the End Of Frame flag. Special circuitry in the receiver must inform the processor of the boundary between the end of the information field and the beginning of the CRC when the information field is not an integral number of n-bit characters. The MPSC^2 performs all of the above functions necessary to implement Bit-Oriented Protocols.

3.5 Functional Description

The MPSC^2 provides to complete serial communications controllers in a single package implementing the following functions:

Parallel to Serial and Serial to Parallel data conversion.

Buffering of outgoing and incoming data, allowing the processor time to respond.

Insertion and deletion of framing bits and characters.

Calculation and checking of Parity and CRC error checking.

Informing the processor when and what action needs to be taken.

Interfacing with the outside world over discrete modem control lines.

The MPSC^2 can be logically divided into the following functional groups (Figure 3-6):

Two identical serial I/O controller channels, each consisting of a Transmitter section and a Receiver section, and

a common Processor Interface that connects the MPSC^2 with the host processor and provides overall device control.

3.6 Transmitter

The MPSC² Transmitter performs all the functions necessary to convert parallel data to the appropriate serial bit streams required by vaious protocols. The major components of the transmitter are shown in Figure 3-7. Control and Status Register fields pertinent to the operation of the Transmitter are summarized in Table 3-1

See Last Page 3.44

Table 3-1 Control and Status Register fields

The primary data flow through the Transmitter begins at the Internal Data Bus. There, characters written to the MPSC² are placed in the Buffer Register. When any character present in the Shift Register has been transferred out, or if the Shift Register was empty, the contents of the Bufffer Register are transferred to the Shift Register and output with the least significant vit first. Then, a Transmitter Buffer Becoming Empty indication (flag) is given. This Double Buffering allows the processor one full character time from this flag to respond with the next character without interrupting data transmission. You should note that it is the transfer of a character from the Data Buffer to the Shift Register rather than the empty condition itself that causes the Transmitter Buffer Becoming Empty indication. At initialization or after a Reset Transmitter Interript/DMA Pending Command is issued to Control Register 0 (CR0) you must write one character to the buffer to reset this flag. The Transmitter Buffer Empty bit in status Register 0 (SR0), always relfects the presence or absence of a character in the buffer.

After a hardware or software reset, the Transmitter Data Output (TxD) is in high (marking) state. You can pull TxD low (spacing) any time by setting the Send Break bit (CR5 Bit 4). TxD remains low until the Send Break bit is reset and any data currently being trensmitted is detroyed.

You can change the number of bits transmitted for each character at any time by modifying the bits/char field (CR5, D -D) before you load the character into the Buffer.

The rate at which data is shifted out is determined by the Transmitter Clock Input (TxC) and the clock mode field (CR4 Bits 6-7). You can select a Clock divisor so that the Data Clock (TxC) rate is equal to 1×, 16×, 32×, or 64× the actual data rate. This field also controls the Receiver Clock and must be set to 1× for synchronous modes (see Asynchronous Reception Section 4.2.2 for use in Asynchronous mode). Each new bit is shifted out on the falling edge of TxC.

The following is a general descussion of the operation of the MPSC² in various protocol modes. For a detailed description of the Registers and examples, see the section *Programming the* MPSC².

3.6.1 Asynchronous Mode

After you select Asynchronous mode, initialize the various parameters (number of bits/character, numver of stop bits, etc.) and enable the Transmitter (CR5 bit 3 eq 1). TxD remains in the high (marking) state. When the first character is written to the Data Buffer, it is transfered to the Shift Register and the Transmitter Buffer Becoming Empty flag is set. A Parity bit, if enabled, and the specified number of Stop Bits (1, 1 1/2 or 2) are apended to the character. The Character plus the start bit are shifted out serially through a one-bit delay. After the character has been completely sent, the next character is loaded into the Shift Register and the process continues. When no more characters are available, TxD remains high and the All Async Character Sent flag (SR1 bit 0) is set until the next character is loaded. The transmitter may be disabled at any time (CR5 bit 3 eq 0) however, transmission of the character currently being sent, if any, is completed. Disabling the transmitter does not reset Transmitter Buffer Becoming Empty or any resultant interrupts or DMA requests. You can clear this flag either by writing a character to the Data Buffer for later transmission or by issuing a Reset Transmitter Interrupt/DMA Pending Command.

The modem control output RTS (Request To Send) may be set or reset at any time with CR5 bit 1. RTS immediately goes to the active state (low) when this bit is set. When reset, RTS does not go high until the Shift Register and the Data Buffer are empty.

The function of the modem control input, CTS (Clear To Send), depends upon the Auto Enables Control (CR3 bit 5). When Auto Enables is reset, any transition of CTS sets the External/Status Change flag but has no affect upon transmission. When Auto Enables is set, character transmission cannot begin until CTS goes low. If CTS goes high, any character currently being transmitted is completed and the transmitter is then disabled until CTS again goes low. The CTS flag, SR0 bit 5, reflects the inverted state of the external CTS pins; that is CTS flag eq 1 when CTS eq low.

3.6.2 COP Synchronous Modes

The MPSC² gives you three distinct COP operating modes: Monosync (8-bit sync character), Bisync (16-bit sync character), and External Sync (the Transmitter operates in the same manner as Monosync). When Bisync mode is selected, you should program the eight least significant bits (first byte) of the sync character into CR6 and the eight most significant bits (second byte) into CR7. For Monowync and External Sync modes you should program CR6 with the 8-bit sync character.

During operation in COP modes, the MPSC² transmitter may be in any one or the following phases:

Disabled Phase: Transmitter Enable is off (CR5, D3 = 0) or CTS is low when the Auto Enables function is used;

Idle Phase: Sync characters are being sent;

Data Phase: Data from the processor is being transmitted;

CRC Phase: (if CRC is used) when the CRC check characters are being transmitted.

After selecting the desired protocol and initializing parameters, the transmitter enters and remains in the Disabled Phase, with TxD high until the Transmitter Enable bit is set. once this is done the transmitter enters the Idle phase, transmits the first sync character and continues to send sync characters until a character is written into the transmit buffer. When the first data character is loaded into the Data Buffer and the current sync character has been sent, the Transmitter enter Data Phase and sends data characters while setting the Transmitter Buffer Becoming Empty flag each time it is ready for the next character.

During the Data Phase, the transmitter may run out of data to send for one of two reasons: (1) The processor is busy and is not able to provide the next data characters within a message, or (2) the data portion of the message is complete and it is time to enter the CRC phase (or the idle phase if CRCis not used). The MPSC^2 automatically handles both of these conditions through a mechanism called the Idle/CRC Latch, the state of which may be read from SR0 D_6.

When the Transmitter is initialized the Idle/CRC Latch is set, indicating that the Transmitter will enter the idle Phase and begin sending sync characters when there is no data to send. Entering this phase aldo sets the Transmitter Buffer Becoming Empty flag (if not already set) to indicate with SR0 D_6 eq 1, that the Idle Phase has been entered.

However, if you reset the Idle/CRC latch with a Reset Idle/CRClatch command to CR0, a lack of data causes the MPSC^2 to enter the CRC phase and begin sending the 16-bit CRC character calculated up to that point. Entering the CRC Phase sets the Idle/CRC Latch which, in turn, sets the External/Status Change flag indicating that the MPSC^2 is sending CRC. Agter you reset the flag, you may send the next data character to the Transmitter and it will be sent immediately following the CRC, or you may do nothing. In either case, the Idle/CRC latch is now set again so the Transmitter enters the idle phase when no further data is available.

You can disable the transmitter during any phase of operation. If the Transmitter is disabled during the Idle or Data Phases the MPSC^2 finishes sending the current character and goes to the Disabled Phase (TxD high). If disabled during the CRC phase, a 16-bit CRC is sent, however, the remainder of the CRC is supplanted by sync with bit positions matching.

The CRC Generator may be programmed to either of two polynomials: CRC-16 ($x^{16}+x^{15}+x^2+1$) or CRC-CITT ($x^{16}+x^{12}+x^5+1$). The CRC Generator may be reset to 0 at any time by issuing a Reset CRC Generator Command to CR0. Since it is sometimes necessary to exclude certain characters from the CRC calculation, the MPSC^2 features a CRC enable/disable control (CR5 D_0) that may be changed just prior to loading a character into the Transmitter buffer to include or exclude that and subsquent characters in the CRC calculation.

3.6.3 SDLC (/HDLC BOP Synchronous) Mode

In SDLC mode, the MPSC^2 Transmitter operates similarly to monosync transmission with te following exceptions:

WR6 is not used for the Transmitter sync character. SDLC flags (sync) are generated internally.

Data and CRC are passed through zero insertion logic before transmission. This logic inserts a 0 bit after transmitting 5 contiguous ones to distinguish information from framing flags.

A special Send SDLC Abort Command is available in CR0. Issuing this command causes at least 8 but less than 14 ones to be transmitted, destroying any data in the Transmitter Shift Register and Buffer. After sending the Abort, the Transmitter enters Idle Phase.

Resetting the CRC generator initializes it to all ones rather than zero and the result bits are inverted before transmission.

3.7 Receiver

The MPSC² Receiver reverses the process performed by the transmitter. It converts the serial data stream of the various protocols back to parallel data for the processor. The major components of the Receiver are shown in Figure 3-8. Control and Status Registers pertinent to the operation of the Receiver are summarized in Table 3-2.

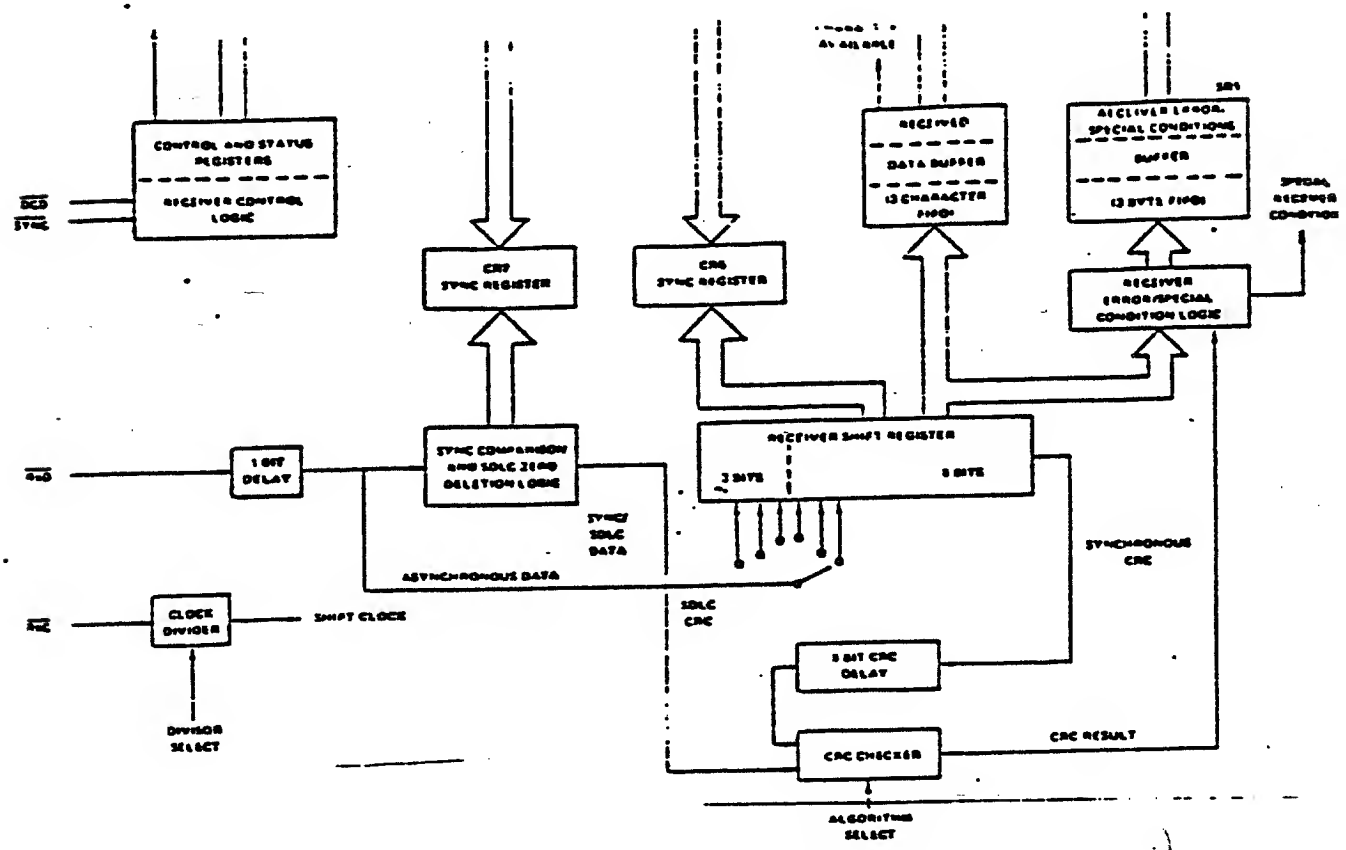

Figure 3-8 Block Diagram of MPSC² Receiver

CONTROL REGISTER	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	CRC CONTROL		COMMAND			REGISTER POINTER		
1				Receiver Interrupt Control				Ext/Status Interrupt Enable
3	Bits/Char		Auto Enables	Enter Sync Hunt Phase	Receiver CRC Enable	SDLC Address Search mode	Sync Char Load Inhibit	Receiver Enables
4	Clock Mode		Sync Format Select		Sync/Async Mode Select		Parity Control	
5						CRC Type		
6	SYNC 1							
7	SYNC 2							

STATUS REGISTER	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	Break/ Abort			Sync/Hunt Mode	DCD			Received Character Available
1	SDLC End of Frame	CRC/ Framing Error	Receiver Overrun Error	Parity Error	SDLC I-Field Residue Code			

Table 3-2 Receiver Control and Status Registers

The primary data path through the receiver begins at the Receiver Data Input RxD. Data passes through a two-bit time delay and into the Receiver Shift Register (the sync data path is descrived later). The point of entry into the Shift Register and hence the number of bits per character is determined by the mode of operation and the Bits/Character field of CR3 (D_6–D_7). You can change this field at any time provided that the character that is currently being assembled has not yet reached the new number of Bits/Character. If the number of Bits/Character specified is less than eight, the character appears right-justified in the Data Buffer (with the parity bit, if parity is enabled) and the left side is filled with ones (see Figure 4.3).

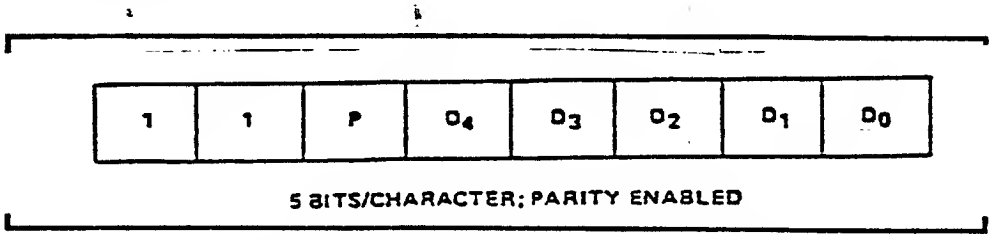

Figure 3-8 data Format Example for Less than 8 Bits/Character

Once the character has been assembled in the Shift Register, it is passed to a three-character First In-First Out buffer (FIFO) and the Received Character Available flag (and SR0 D_0) is set to inform the processor that a character is available. The three-character buffer allows the processor upt to four character times to service the Receiver without losing data. This feature enchances data reliability at high speeds while relaxing software timing requirements. The Received Character Available flag is reset when all characters in the buffer have been read, i.e., the buffer is empty.

As each character is transferred to the Buffer, it is checked for errors or special conditions and that information is placed in a parallel FIFO Error Buffer so that the status associated with each character can be read with that character through Status Register 1. Reading a character from the Data Buffer moves the next character and its status to the top of the FIFO. You should read the status first, if it is of interest, and then the data.

The rate at which data is shifted into the Receiver is controlled by the Receiver Clock Input ($\overline{RxC}$) and the clock mode field (CR4 D_6–D_7). This field also controls the Transmitter clock mode. In any of the synchronous modes, you must select the 1x clock mode. In asynchronous mode you may select a divisor such that the clock rate ($\overline{RxC}$) equals 1×, 16×, 32×, or 64× the actual data rate. However, if you select the 1×mode, the clock must be externally synchronized with the data (see Sec. 4.1.3). RxD is always sampled on the rising edge of $\overline{RxC}$.

The Data Carrier Detect ($\overline{DCD}$) Input works the same way as $\overline{CTS}$ except that it enables the receiver when Auto Enables is set.

3.7.1 Asynchronous Mode

After initializing and enabling the MPSC² Receiver, the receiver logic begins sampling the RxD input for a high-to-low (marking -to-spacing) transition on each rising edge of TxD. When the transition is found, the receiver waits 1/2 bit time, (for example, eight clock periods if the clock mode is 16×) and samples again to ensure that RxD is still low, improving the MPSC²'s noise immunity. If RxD is still low, the MPSC² assumes this is the middle of the start bit and 1 bit time later begins to sample RxD to assemble the required number of data and parity (if enabled) bits.

Once the character is assembled, the MPSC² waits one more bit time and again samples RxD. If RxD is not high, the stop bit is missing and a Framing Error is indicated when the character is passed to the Data Buffer. If a Framing Error has occurred, the MPSC² receiver waits 1/2 bit time before beginning to sample again to avoid interpreting the Framing Error as a new start bit.

Note that in the 1× Clock Mode, the Receiver simply waits one clock period after the first high-to-low transition is detected and then begins assembling the character. It is for this reason that data and clock must be synchronized in this mode.

The Break/ABort bit, D_7 of SR0 is set when a null character plus Framing Error is detected (i.e.RxD is low for more than one full character time). Break detection also sets the External/Status Change flag. When RxD returns high and the break has ended, D_7 is reset to 0 and the External Status Change is once again set. After the break, a single null character is present in the Data Buffer. It should be read and discarded.

The following errors may occur during operation and are flagged in Status Register 1:

Framing Error — See above discussion.

Parity error — If parity is enabled and a parity error occurs, the Parity Error bit D_4 is set. Once a Parity Error has occurred, the Parity Error bit remains set for subsequent characters until reset by an Error Reset command to CR0. You need only check the end of a message or block to determine if a Parity Error had occurred.

Overrun Error — If the Data Buffer is full with three characters, and a fourth character is received, the last character in the Buffer is overwritten and the Overrun Error bit D_5 is set. Like Parity Error, Overrun Error remains set until the Error Reset command is issued.

3.7.2 COP Synchronous Modes

The MPSC² gives you three distinct COP Operation modes: (1) Monosync (8-bit sync character), (2) Bisync (16-bit character), and (3)External Sync (the $\overline{\text{SYNC}}$ pin is used as an input to inform the MPSC² that suunchronization has been achieved extemally). When Monosync Mode is Selected, CR7 should be programmed with the 8-bit sync character to be matched by the Receiver. In Bisync Mode CR6 should contain the least significant bits (first byte) and CR7 should contain the most significant bits (second byte) of the 16-bit character ro be matched. In External Sync mode, no sync character is required by the Receiver.

During operation in the COP modes, the MPSC² receiver is in one of two phases: (1) Sync Hunt Phase or (2) Data Phase. The Receiver automatically enters Sync Hunt Phase when it is enabled (CR3, D_0).

In Monosync Mode, the incoming data stream passes through and is compared to the sync character in CR7. When a match is found, the receiver switches to the Data Phase and begins to pass data to the shift register. If you determine at any time that synchronization has been lost, you may re-enter the Sync Hunt phase by setting the Enter Hunt Phase bit (D_4) in CR3. When the Hunt Phase is entered or left, the External/Status change flag is set. When SR0 D_4 (Sync/Hunt) eq one, it indicates that the Receiver is in Hunt Phase.

Operation is similar in Bisync Mode, however, when a match is found, CR6 is also checked against the shift register contents and the Hunt Phase is left only if the bytes match. In both Monosync and Bisync Modes, the SYNC pin is used as an output which goes momentarily low any time a sync pattern is detected whether the receiver is in the Hunt or Data Phase. See Figure 2.3 for a detailed timing diagram.

You can inhibit the transfer of sync characters to the Data Register by setting the Sync Char Load Inhibit bit (CR3, D_1). Since the CRC calculation on sync is not inhibited by this bit, you should use it only to strip leading sync characters from a message if you are using CRC Block Check.

Because of the 8-bit delay between the shift register and the CRC Checker, CRC status (SR1, D_XXX) is not valid immediately after the CRC character is received. CRC status is valid 16-bit times after the last CRC character is transferred to the Receive Buffer, or 20 bit times after the last CRC bit us shifted in at RxD.

3.7.3 SDLC (/HDLC BOP Synchronous) Mode

The MPSC² provides you with high-level processing capability for handling Bit-Oriented Protocols. When you select SDLC Mode, CR7 must be programmed with the SDLC Flag character 01111110.

When operating in SDLC mode, the Receiver can be in one of three phases: Hunt Phase, Address Search Phase, or Data Phase.

The receiver automatically enters Hunt Phase When first enabled. The incoming data stream passes through the one-bit delay and enters the Sync Comparison/Zero Deletion Logic where the following three operations are performed:

First, whenever a 0 bit follows five consecutive ones, that 0 is deleted from the data stream. Second, if six consecutive ones are received, a Flag Character Received indication is given internally. Third, if eight or more ones are received, an Abort is indicated and the External/Status Change Flag is set. Flags and Aborts are not transferred to the Receiver Shift Register.

Once a Flag is detected, the Receiver leaves the Hunt Phase (setting the External/Status Change Flag) and, if Address Search Mode (CR3-D_2) is enabled, it enters Address Search Phase. Once this phase

is entered, the MPSC² Receiver compares the first 8-bit non-flag character to the contents of Control Register 6. If the two values match, or the received characteris the Global Address 11111111, the Receiver immediately enters Data Phase and character assembly begins with this character. If no match is found and the value is not the Global Address, the Receiver remains in Address Search Phase and no data characters are assembled until a Flag followed by the correct address is encountered. If Address Search Mode is not enabled, Data Phase is entered immediately and character assembly begins with the first non-flag character. Since all messages are framed with flag characters, you can skip an incoming message at any time simply by setting the Enter Hunt Phase bit (D_4) in CR3.

Once in Data Phase, characters are assembled according to the number of bits or character specified until the next End of Frame flag is encountered. The Receiver then sets the Special Receive Condition flag and transfers the character currently being assembled to the Receiver Buffer regardlessof the number of bits actually assembled. A special Residue Code placed in the Status Buffer (SR1) uses the number of bits assembled to indicate the boundary between the data and CRC characters (see Section 5.1 for a more detailed description of the Residue code). If Address Search Mode is enobled, the Receiver once again enters Address Search Phase.

Unlike the COP Mode of operation, data from the Sync Comparison/Zero Deletion Logic passes directly to the CRC checker. As a result, when the End of Frame Flag is detected, the CRC calculation is complete and the error status is passed to the Status Buffer along with the residue code. The CRC checker is automatically reset to all ones at this time.

3.8 Bus Interface Controller

The Bus Interface Controller is the interface between the Transmitter and Receiver sections and the Processor bus. The major components of this section are shown in Figure 3-10. Thw Control and status registers pertinent to the operation to the Control Section are illustrated in Table 3-3.

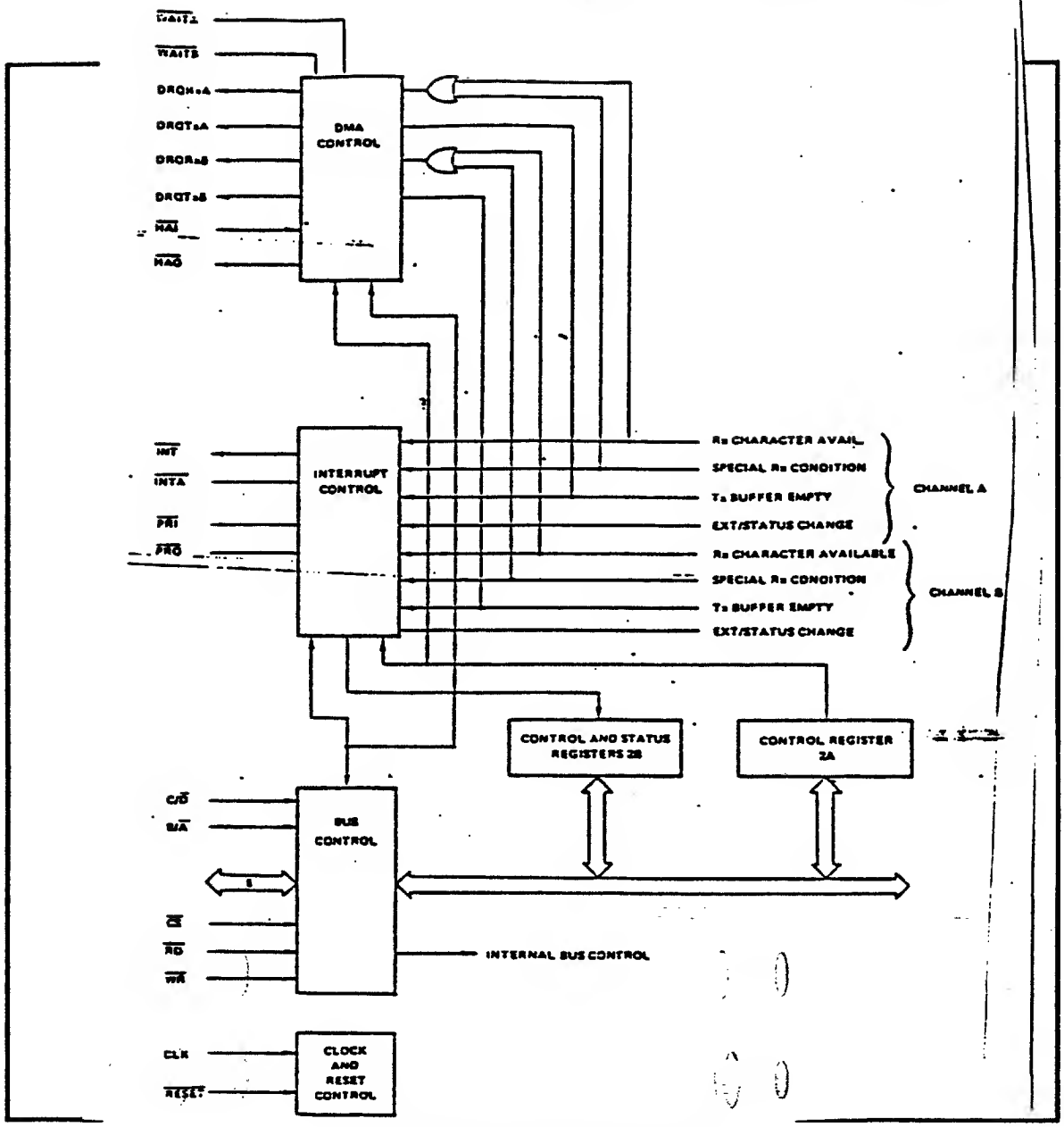

Figure 3-10 Bus Interface Controller

$\overline{CS}$	B/$\overline{A}$	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	OPERATION
1	X	X	X	X	No operation, the MPSC² is deselected
0	X	X	1	1	No operation, the MPSC² is deselected
0	0	0	1	0	Write a char to Channel A transmitter
0	0	0	0	1	Read a char from Channel A recevier
0	0	1	1	0	Write a control byte to Channel A
0	0	1	0	1	Read a status byte from Channel A
0	1	0	1	0	Write a char to Channel B transmitter
0	1	0	0	1	Read a char from Channel B receiver
0	1	1	1	0	Write a control byte to Channel B
0	1	1	0	1	Read a status byte from Channel B
0	X	X	0	0	Illegal

Table 3-3 Read/Write Selection

The Bus Interface Controller can be divided into four major components:

Bus Control Logic
Interrupt Control Logic
DMA Control Logic
Clock and Reset Control Logic

All of these components interact to provide a flexible high-performance interface between the bus architecture defined by your processor and application and the various internal elements that make up the MPSC².

3.8.1 Bus Control Logic

The Bus Control Logic determines the direction and internal source or destination of data and control transfers between the MPSC² and the peocessor bus. During operation of the MPSC², the Bus Control Logic may operate in any of three distinct modes: Processor Read/Write, Interrupt Acknowledge, and DMA Cycle. These last two modes are described in detail in Sections 4.3.2 and 4.3.3.

Processor Read/Write mode is the normal mode of operation. The processor transfers data or commands and status to or from the MPSC² with its instruction set. The MPSC² is enabled for Processor Read/Write Mode when the Chip Select (CS) input is made active (low). The direction of the transfer is controlled by enabling either the Read (RD) or Write (WR) inputs. The B/A input determines the source/destinatin channel for the transfer and the C/D input specifies whether the transfer is character data or control/status information. These inputs are generally connected to the two low-order address lines.

3.8.2 Interrupt Control Logic

The Interrupt Control Logic performs two functions: it prioritizes various internal input requests, and places the appropriate information on the data bus during an Interrupt Acknowledge cycle (if you enabled the MPSC²'s vectored interrupt feature).

Each MPSC² channel can generate four different types of interrupt requests:

Received Character Available
Special Received Condition (character received but with an error or SDLC End of Frame flag received)
Transmitter Buffer has Become Empty
External input (CTS, DCD, SYNC, Internal Status (Sync, Idle/CRC Latch) Change)

When any of these requests occur, the Interrupt Control logic determines whether to accept the request at that time, issue an interrupt request by setting the INT output low when the request is accepted, and, if Vectored Interrupt mode is enabled, place the interrupt information on the data bus during the times that the Interrupt Acknowledge input (INTA) is activated by the processor.

As an example, assume that the Channel A DCD input has just changed state causing an External/Status Change interrupt request. The following sequence occurs:

If all the following conditions are true:

External/Status Change Interrupts are enabled
No higher priority interrupt requests are pending
PRI is active, and
The MPSC² is not acknowledging a pending lower priority interrupt request

then:

The interrupt control logic accepts the interrupt request and sets INT active and PRO inactive.

If Vectored Interrupt mode is enabled, the MPSC² may place information on the data bus in response to a series of INTA pulses as shown in the chart below:

Table 3-4 Vectored Interrupt Mode

When operating in the 8080/5 modes, the MPSC^2 issues an 8080-type CALL (CD_{16}) instruction where vv is the contents of Control Register 2B (modified by the cause of the interrupt if the Status Affects Vector feature is enabled). In particular, and MPSC^2 programmed for 8085 Master mode ALWAYS places the CALL opcode on the data bus regardless of whether that MPSC^2 has a pending interrupt request. To avoid problems caused by momentary bus contention, you should never program more than one device, MPSC^2 or other, to operate in this mode.

In 8086 mode, The MPSC^2 places the Vector on the data bus during the second Interrupt Acknowledge To Vector the processor to the approximate location in low memory.

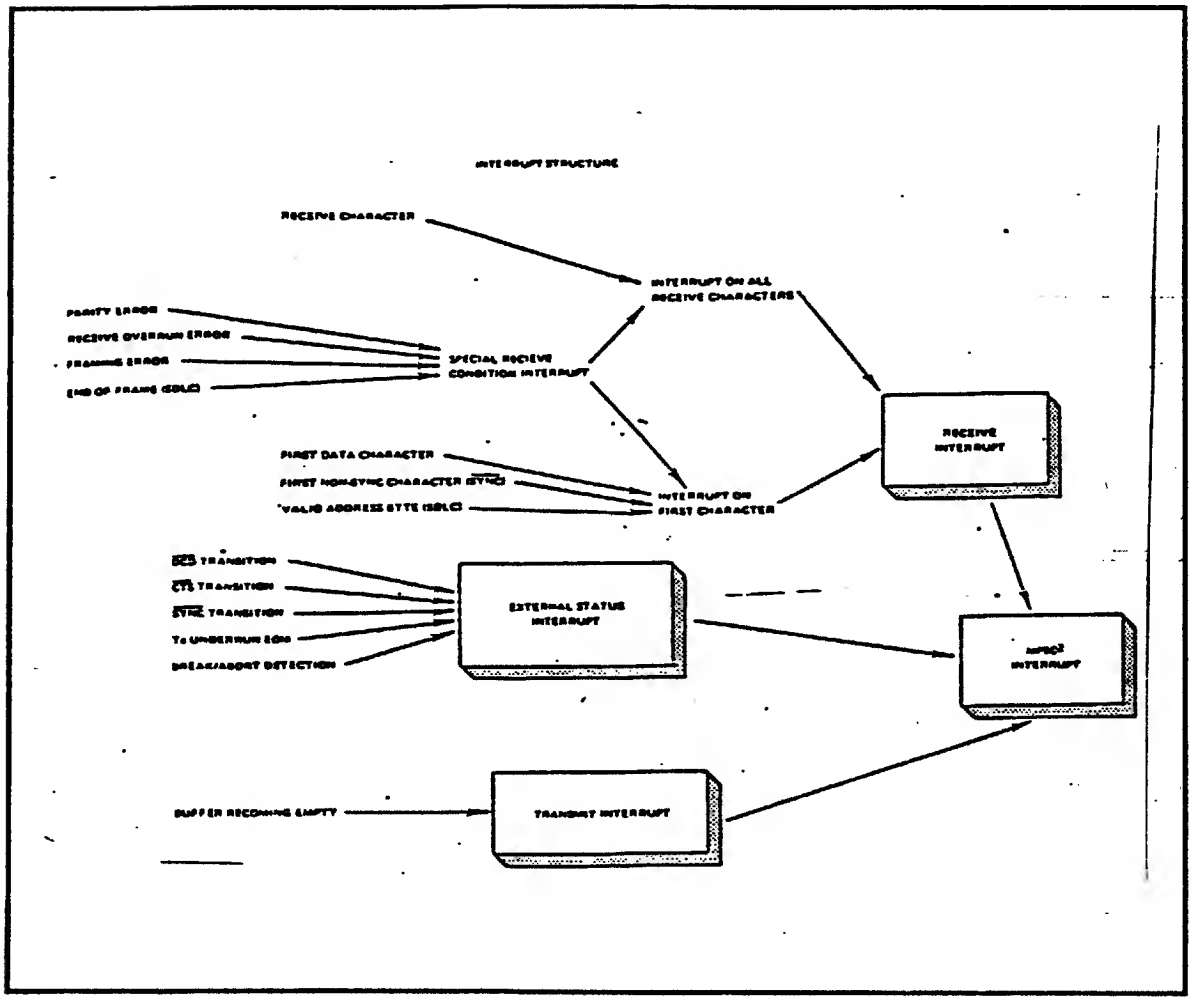

Figure xxxxx MPSC² Interrupt Conditions

Figure 4.7 illustrates the action of the Interrupt Control Logic during an Interrupt Acknowledge Sequence.

At the beginning of the first Interrupt Acknowledge cycle, the interrupt prioritization logic is frozen to permit any late interrupt requests by higher priority devices to ripple through and resolve internal priorities before the second interrupt pulse.

At the end of the second INTA pulse, the INT output is released by the acknowledging device and the interrupt prioritization logic is re-enabled with an Interrupt in Service flag set. As long as this flag is set, PRO is held high and only internal interrupt requests with a priority higher than the one currently being serviced are accepted.

While the interrupt is being serviced, the processor issues an End of Interrupt (EOI) command to the MPSC 2 to reset the Interrupt Control Logic to its previous state. This scheme permits nested interrupts to be serviced and the priority daisy chain to be properly maintaned.

When the MPSC 2 is operated in Non-Vectored Interrupt mode, the Interrupt Control Logic operates in a similar manner except that INTA is not used and no vector information is placed on the data bus. Rather, the Interrupt Acknowledge sequence is simulated by reading the Vector (modified if Status Affects Vector is enabled) in Status Register 2B.

3.9 Programming The MPSC²

The software operation of the MPSC² is very straightforward. Its consistent register organization and high-level command structure help to minimize the number of operations required to implement complex protocol designs. Programming is further simplified by the MPSC²'s extensive interrupt and status reporting capabilities.

This section is divided into two parts. The first is a detailed description of the commands, bits, and fields in the various MPSC² Control ansd Status Registers. The second part provides programming examples and flowcharts for the MPSC²'s various operating modes to assist you in developing software for your specific application.

3.9.1 The MPSC² Registers

The MPSC² interfaces to the system software with a number of Control and Status Registers associated with each channel. Commonly used commands and status bits are access directly through Control and Status Registers 0. Other functions are accessed indirectly with a register pionter to minimize the address space that must be dedicated to the MPSC².

Control Register	Function
0	Frequently used commands and register pointer control
1	Interrupt control
2	Processor/bus interface control
3	Receiver control
4	Mode Control
5	Transmitter control
6	Sync/address character
7	Sync character

Control Registers

Status Registers	Function
0	Buffer and "External/Status" Status
1	Received character error and special condition status
2 (Channel B only)	Interrupt Vector

Status Registers

All Control and Status Registers except number 2 are separately maintained for each channel. Control and Status Registers 2 are linked with the overall operation of the MPSC² and have different meanings when addressed through different channels.

When initializing the MPSC², Control Register 2A (and 2B if desired) should be programmed first to establish the MPSC² Processor/Bus interface mode. You may then program each channel to be used separately, beginning with Control Register 4 to set the protocol mode for that channel. The remaining registers may then be programmed in any order.

3.9.2 Control Register 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CRC Control Commmand		Command			Register Pointer		

Figure 3-1 Control Register 0

Register Pointer (D_0–D_1)

The Register Pointer specifies which Register number is accessed at the next Control Register Write or Status Register Read. After a hardware or software Reset, the Register pointer is set to 0. Therefore, the first control byte goes to control register 0. When the Register Pointer is set to a value other than 0, the next conrol or status (C/D =1) access is to the specified register, after which the Pointer is reset to 0. You can freely combine other commands in Control Register 0 with setting the Register Pointer.

Command (D_3–D_5)

Commands commonly used during the operation of the MPSC² are grouped in Control Register 0 for your convenience. They are:

Null (000)

This command has no effect and is used when you wish to set only the Register Pointer or issue a CRC Command.

Send Abort (001)

When operating in SDLC Mode, this command causes the MPSC² to transmit the SDLC Abort code, issuing 8 to 13 consecutive ones. Any data currently in the Transmitter or the Transmitter Buffer is destroyed After sending the Abort, the Transmitter reverts to the Idle Phase (flags).

Reset External/Status Interrupts (010)

When the External/Status Change flag is set, the condition bits D -D of Status Regiŝter 0 are latched to allow you to capture short pulses that may occur. The Reset External/Status Interrupts Command clears a pending interrupt and re-enables the latches so that new interrupts may be sensed.

Channel Reset (011)

This command has the same effect on a single channel as an external Reset at pin 2. A Channel Reset command to channel A resets the internal interrupt prioritization logic. This Does not occur when you issue a Channel Reset command to channel B. You must reinitialize all control registers associated with the channel that you reset. After a channel reset, you must wait at least four system clock cycles before writing new commands or controls to that channel.

Enable Interrupt on Next Character (100)

When operating the MPSC² in Interrupt on First Received Character mode, you may issue this command at any time (generraly at the end of a message), to re-enable the interrupt logic for the next received character.

Reset pending Transmitter Interrupt/DMA Request (101)

You can reset a pending Transmitter Buffer Becoming Empty interrupt or DMA Request without sending another character by issuing this command (typically at the end of a message). A new Transmitter Buffer Becoming Empty interrupt or DMA request is not made until another character has been loaded and transferred to the Transmitter Shift Register or when, if operating in Synchronous or SDLC mode, the CRC character has been completely sent and the first wync or flag character loaded into the Transmitter Shift Register.

Error Reset (110)

This command resets a Special Receive Condition interrupt. It also re-enables the Parity and Overrun Error Latches that allow you to check for these errors at the end of a message.

End of Interrupt (111) (Channel A only)

Once an interrupt request has been issued by the MPSC², all lower priority internal and external interrupts in the daisy chain are held off to permit the current interrupt to be serviced while allowing higher priority interrupts to occur. At some point in your interrupt service routine (generally at the end), you must issue the End of Interrupt command to channel A to re-enable the daisy chain and allow any pending lower priority internal interrupt requests to occur.

CRC Control Commands (D_6 - D_7)

These commands control the operation of the CRC generator/checker logic.

Null (00)

This command has no effect and is used when issuing other commands or setting the Register Pointer.

Reset Receiver CRC Checker (01)

This command resets the CRC Checker to 0 when the channel is in a Synchronous mode and resets to all ones when in SDLC mode.

Reset Idle/CRC latch (11)

This command resets the Idle/CRC Latch so that when a transmitter underrun condition occurs (that is, the transmitter had no more characters to send), the transmitter enters the CRC phase of operation and begins to send the 16-bit CRC character calculated up to that point. The latch is then set so that if the underrun condition persists, idle characters are sent following the CRC. After a hardware or software reset, the latch is in the set state.

3.9.3 Control Register 1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Wait Function Enable	ø	Wait on Receiver Transmitter	Receiver Interrupt Mode		Condition Affects Vector	Transmitter Interrupt Enable	Ext/Status Interrupt Enable

Figure 3-2 Control Register 1

External/Status Interrupt Enable (D_0)

When this bit is set to one, the MPSC² issues an interrupt whenever any of the following occur:

transition of $\overline{DCD}$ input
transition of $\overline{CTS}$ input
transition of $\overline{SYNC}$ input
entering of leaving synchronous Hunt Phase break detection or termination
SDLC abort detection of termination
Idle/CRC latch becoming set (CRC being sent)

Transmitter Interrupt Enable (D_1)

When this bit is set to one, the MPSC² issues an interrupt when:

the character currently in the transmitter buffer is transferred to the Shift Register (Transmitter Buffer Becoming Empty) or,

the transmitter enters Idle Phase and begins transmitting sync or flag characters.

Status Affects Vector (D_2)

When this bit is set to 0 and MPSC² is in Vectored Interrupt Mode (see CR2A), the interrupt vector (and SR2B) is the value that you placed into control register 2B during MPSC² initialization. If Status Affects Vector is set to 1, the vector is modified to reflect the condition that caused the interrupt. See Section 4.3.2 for a detailed explanation of the MPSC²'s vectored interrupt feature.

Receiver Interrupt Mode (D_3–D_4)

This field controls how the MPSC²'s interrupt/DMA logic handles the Character Received Condition.

Receiver Interrupts/DMA Request Disabled (00)

The MPSC² does not issue an interrupt or a DMA request when a character has been received.

Interrupt on First Received Character Only (01)

(and issue a DMA Request)

In this mode, the MPSC² issues an interrupt only for the first character received after an Enable Interrupt on First Character command (CR0) has been given. If the Channel is in DMA mode, a DMA request is issued for each character received including the first. This mode is generally used when using the MPSC² in DMA or Block Transfer Mode to signal the processor that the beginning of an incoming message has been received.

Interrupt On All Received Characters (10)

(and issue a DMA Request) Parity Error Is a Special Receive Condition

In this mode, an interrupt (and DMA Request if DMA mode is selected) is issued whenever there is a character present in the Receiver Buffer. A Parity Error is considered a Special Receive condition.

Interrupt On All Received Characters (11)

(and issue a DMA request) Parity Error Is Not a Special Receive Condition

This mode is the same as above except that a Parity Error is not considered a Special Receive Condition.

The following are considered Special Receive Conditionsah and, when Status Affects Vector is enabled, cause and interrupt vector different from that caused by a Received Character Available condition:

Receiver Overrun Error
Asynchronous Framing Error
Parity Error (if specified)
SDLC End of Message (Final Flag received)

Wait on Receiver/Transmitter (D_5)

If the Wait function is enabled for Block Mode transfers, setting this bit to 0 causes the MPSC² to issue a wait (WAIT output goes low) when the processor attempts to write a character to the Transmitter while the Transmitter Buffer is full. Setting this bit to 1 causes the MPSC² to issue a Wait when the processor attempts to read a character from the Receiver while the Receiver Buffer is empty.

Wait Function Enable (D_7)

Setting this bit to 1 enables the Wait function as described above and in Section 4.3.3.

3.9.4 Control Register 2 (Channel A)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
$\overline{\text{SYNCB}}$/ RTSB	ø	Interrupt Vector Mode			Priority	DMA Mode Select	

Figure 3-2 Control Register 2 (Channel A)

DMA Mode Select (D_0–D_1)

Setting this field establishes whether channels A and B are used in DMA mode (i.e. data transfers are performed by a DMA controller) or in non-DMA mode where transfers are performed by the processor in either Polled, Interrupt, or Block Transfer Modes. The function of some MPSC² pins are also controlled by this field.

		Channel		Pin Function					
D_1	D_0	A	B	11	26	29	30	31	32
0	0	Non-DMA	Non-DMA	$\overline{\text{WAITB}}$	$\overline{\text{DTRB}}$	$\overline{\text{PRI}}$	$\overline{\text{PRO}}$	$\overline{\text{DTRA}}$	$\overline{\text{WAITA}}$
0	1	DMA	Non-DMA	DRQTxA	$\overline{\text{HAI}}$	$\overline{\text{PRI}}$	$\overline{\text{PRO}}$	$\overline{\text{HAO}}$	DRQRxA
1	0	DMA	DMA	DRQTxA	$\overline{\text{HAI}}$	DRQRxB	DRQTxB	$\overline{\text{HAO}}$	DRQRxA
1	1	illegal	-	-	-	-	-	-	-

Table 3-3 DMA Mode Selection

Priority (D_2)

This bit allows you to select the relative priorities of the various interrupt and DMA conditions according to your application.

D_2	Mode CHA	Mode CHB	DMA Priority Relation	Interrupt Priority Relation
0	INT	INT		RxA>TxA>RxB>TxB>ExTA>ExTB
1	INT	INT		RxA>RxB>TxA>TxB>ExTA>ExTB
X	DMA	INT	RxA>TxA	RxA>RxB>TxB>ExTA>ExTB
0	DMA	DMA	RxA>TxA>RxB>TxB	RxA>RxB>ExTA>ExTB
1	DMA	DMA	RxA>RxB>TxA>TxB	RxA>RxB>ExTA>ExTB

Table 3-4 DMA/Interrupt Priorities

Interrupt Vector Mode (D_3–D_5)

This field determines how the MPSC² responds to an Interrupt Acknowledge Sequence from the processor. see Section 4.3.2 for a detailed description of the MPSC² response in these modes.

D_5	D_4	D_3	Mode	Status Register 2B and Interrupt Vector bits affected when Condition Affects Vector is enabled
0	0	0	Non-Vectored	D_4 D_3 D_2
0	0	1	Non-Vectored	D_4 D_3 D_2
0	1	0	Non-Vectored	D_2 D_1 D_0
0	1	01	illegal	—
1	0	0	8085 Master	D_4 D_3 D_2
1	0	1	8085 Slave	D_4 D_3 D_2
1	1	0	8086	D_2 D_1 D_0
1	1	1	illegal	—

Table 3-5 Interrupt Acknowledge Sequence Response

$\overline{\text{SYNCB}}$/$\overline{\text{RTSB}}$ Select (D_7)

Programming a 0 into this bit selects $\overline{\text{RTSB}}$ as the function of pin 10. A one selects $\overline{\text{SYNCB}}$ as the function.

3.9.5 Control Register 2 (Channel B)

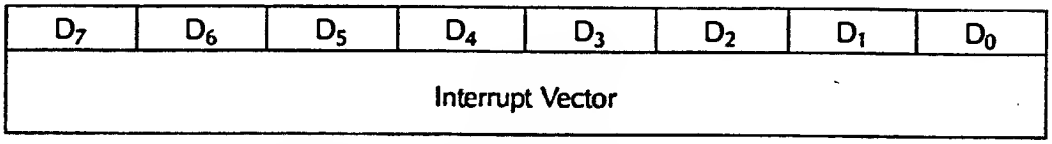

Figure 3-4 Control Register 2 (Channel B)

Interrupt Vector (D_0 - D_7)

When the MPSC² is used in Vectored Interrupt Mode, the contents of this register are placed on the bus during the appropriate portion of the Interrupt Acknowledge Sequence. Its value is modified if Status Affects Vector is enabled. You can read the value of CR2B at any time. This feature is particularly useful in determining the cause of an interrupt when using the MPSC² in Non-vectored Interrupt Mode.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Number of Received Bits/Character		Auto Enables	Enter Hunt Phase	Receiver CRC Enable	Address Search Mode	Sync Character Load Inhibit	Receiver Enable

Figure 3-5 Control Register 3

3.9.6 Control Register 3

Receiver Enable (D_0)

After the Channel has been completely initialized, setting this bit to 1 allows the Receiver to begin operation. You may set this bit to 0 at any time to disable the Receiver.

Sync Character Load Inhibit (D_1)

In a synchronous mode, this bit inhibits the transfer of sync characters to the Receiver Buffer, thus performing a "sync stripping" operation. When using the MPSC²'s CRC checking ability, you should use this feature only to strip leading sync characters preceeding a message sincae the Load Inhibit does not exclude sync characters embedded in the message from the CRC calculation. Synchronous Protocols using other types of block checking such as checksum or LRC are free to strip embedded sync characters with this bit.

Address Search Mode (D_6)

In SDLC Mode, setting this bit places the MPSC² in Address Search Mode where character assembly does not begin until the 8-bit character (secondary address field) following the starting flag of a message matches either the address programmed into CR6 or the Global Address 1111111.

Receiver CRC Enable (D_3)

This bit enables and disables (1 = enable) the CRC checker to allow you to selectively include or exclude characters from the CRC calculation. Ths MPSC² fratures a one-character delay between the Receiver Shift Register and the CRC checker so that the enabling or disabling takes effect with the last character transferred from the Shift Register to the Receiver buffer. You therefore have one full character time in which to read the character and decide whether it should be included in the CRC calculation.

Enter Hunt Phase (D_4)

Although the MPSC² Receive Automatically enters Sync Hunt Phase after a reset, there are times when you may wish to reenter it, such as when you have determined that synchronization has been lost or, in SDLC mode, to ignore the current incoming message. Writing a 1 into this bit at any time after initialization causes the MPSC² to reenter Hunt Phase.

Auto Enables (D_5)

Setting this bit to 1 causes the DCD and CTS inputs to act as enable inputs to the Receiver and Transmitter, respectively.

Number of Received Bits/Character (D_6–D_7)

This field specifies the number of data bits assembled to make each character. You may change the value on the fly while a character is being assembled and ig the change is made before the new number of bits has been reached, it affects that character. Otherwise the new specifications take efect on the next character received.

D_7	D_6	Bits/character
0	0	5
0	1	7
1	0	6
1	1	8

Table 3-6 Receive Bits/Character

3.9.7 Control Register 4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Clock Rate		Sync Mode		Number of Stop Bits Sync Mode		Parity Even/Odd	Parity Enable

Figure 3-6 Control Register 4

Parity Enable (D_0)

Setting this bit to 1 adds an extra data bit containing parity information to eacha transmitted character. Each received character is expected to contain this extra bit and the Receiver Parity Checker is enabled.

Parity Even/Odd (D_1)

Programming a 0 into this bit when parity is enabled causes the transmitted parity bit to take on the value required for Odd Parity. The received character is character for Odd Parity. Conversely, a 1 in this bit signifies Even Parity generation and checking.

Number of Stop Bits/Sync Mode (D_2–D_3)

This field specifies whether the channel is used in Synchronous (or SDLC) Mode or in Asynchronous Mode. In Asynchronous, this field also specifies the number of bit times used as the Stop Bit length by the Transmitter. The Receiver always checks for 1 Stop Bit.

D_3	D_2	Mode
0	0	Synchronous Modes
0	1	Asynchronous 1 Bit time (1 stop bit)
1	0	Asynchronous 1 1/2 Bit times (1 1/2 stop bits)
1	1	Asynchronous 2 bit times (2 stop bits)

Table 3-7 Stop Bit Setting

Sync Mode (D_4–D_5)

When the Number of Stop Bits/Sync Mode field is programmed for synchronous modes (D_2 D_3 = 00), this field specifies the particular synchronous format to be used. This field is ignored in Asynchronous mode.

Sync Mode 1 D_5	Sync Mode 2 D_4	Mode
0	0	8-bit internal synchronization character (monosync)
0	1	16-bit internal synchronization character (bisync)
1	0	SDLC
1	1	external synchronization (sync pin becomes an input)

Table 3-8 Synchronous Formats

Clock Rate (D_6–D_7)

This field specifies the relationship between the Transmitter and Receiver Clock inputs ($\overline{TxC}$, $\overline{RxC}$) and the actual data rate at TxD and RxD. When operating in a synchronous mode you must specify a 1mul clock rate. In Asynchronous modes, any of the rates may be specified, however, with a 1mul Clock Rate the Receiver cannot determine the center of the start bit. In this mode, you must externally synchronize the sampling (rising) edge of $\overline{RxC}$ with the data.

Clock Rate 1 D_7	Clock Rate 2 D_6	Clock Rate
0	0	Clock rate = 1× data rate
0	1	Clock rate = 16× data rate
1	0	Clock rate = 32× data rate
1	1	Clock rate = 64× data rate

Table 3-9 Clock Rates

3.9.8 Control Register 5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
DTR	Number of Transmitted Bits/Character		Send Break	Transmitter Enable	CRC Polynomial Select	$\overline{RTS}$	Transmitter CRC Enable

Figure 3-7 Control Register 5

Transmitter CRC Enable (D_0)

A 1 or a 0 enables or desables, respectively, CRC Generator Calculation. The enable or disable does not take effect until the next character is transferred from the Transmitter Buffer to the Shift Register, thus allowing you to include or excl;ude specific characters before loading the next character, it and subsequent characters are included or excluded from the calculation. If this bit is 0 when the transmitter becomes empty, the MPSC^2 goes to the Idle Phase, regardless of the state of the Idel/CRC latch.

RTS (D_1)

In Synchronous and SDLC modes, setting this bit to 1 causes the RTS pin to go low while a 0 causes it to go high. In Asynchronous mode, setting this bit to 0 does not cause RTS to go high until the Transmitter is completely empty. This feature facilitates programming the MPSC^2 for use with asynchronous modems.

CRC Polynomial Select (D_2)

This bit selects the polynomial used by the transmitter and receiver for CRC generation and checking. A 1 selects the CRC-16 polynomial ($x^{16}+x^{15}+x^2+1$). A 0 selects the CRC-CCITT Polynomial ($x^{16}+x^{12}+x^5+1$). In SDLC mode, you must select CRC-CCITT. You may use either polynomial in other synchronous modes.

Transmitter Enable (D_3)

After a Reset, the transmitted data output (TxD) is held high (marking) and the transmitter is disabled until this bit is set.

In Asynchronous mode, TxD remains high until data is loaded for transmission.

In Synchronous and SDLC 2 modes, the MPSC^2 automatically enters Idle Phase and sends the programmed sync or flag characters.

When the transmitter is disabled in Asynchronous mode, any character currently being sent is completed before TxD returns to the marking state.

If you disable the transmitter during the Data Phase in Synchronous mode, the current character is sent, then TxD goes high (marking).

In SDLC mode, the current character is sent, but the marking line following is zero-inserted. That is, the line goes low for one bit time out of every five.

You should never disable the transmitter during the SDLC Data Phase unless a reset is to follow immediately. In either case, any character in the buffer register is held.

Disabling the transmitter during the CRC Phase causes the remainder of the CRC character to be bit-substituted with sync (or flag). The total number of bits transmitted is correct and TxD goes high after they are sent.

If you disable the transmitter during the Idle Phase, the remainder of the sync (flag) character to be sent, then TxD goes high.

Send Break (D_4)

Setting this bit to 1 immediately forces the Transmitter Output (TxD) low (spacing). This function overrides the normal transmitter output and destroys any data being transmitted although the Tranxmitter is still in operation. Resetting this bit releases thr Transmitter Output.

Transmitted Bits/Character (D_5–D_6)

This field controls the number of data bits transmitte din each character. You may change the number of bits/character by rewriting this field just before you load the first character to use the new specification.

D_6	D_5	Bits per Character
0	0	5 or less (see below)
0	1	7
1	0	6
1	1	8

Table 3-10 Transmit Bits per Character

Normally each character is sent to the MPSC² right-justified and the unused bits are ignored. However, when sending 5 bits or less the data should be formatted as shown below to inform the MPSC² of the precise number of bits to be sent.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Number of Bits/Char
1	1	1	1	0	0	0	D_0	1
1	1	1	0	0	0	D_1	D_0	2
1	1	0	0	0	D_2	D_1	D_0	3
1	0	0	0	D_3	D_2	D_1	D_0	4
0	0	0	D_4	D_3	D_2	D_1	D_0	5

Table 3-11 Transmit Bits per Character for 5 Characters or less

DTR (Data Terminal Ready) (D_7)

When this bit is 1, the DTR output is low (active). Conversely, when this bit is 0, DTR is high.

3.9.9 Control Register 6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync Byte 1							

Figure 3-8 Control Register 6

Sync Byte 1

Sync Byte 1 is used in the following modes:

Monosync: 8-bit sync character transmitted during the Idle Phase

Bisync: Least significant (first) 8 bits of the 16-bit Transmit and Receive sync character

External Sync: Sync character transmitted during the Idle Phase

SDLC: Secondary Address vaue matched to Secondary Address field of the SDLC frame when the MPSC² is in Address Search Mode.

3.9.10 Control Register 7

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync Byte 2							

Figure 3-9 Control Register 7

Sync Byte 2 (D_0–D_7)

Sync Byte 2 is used in the following modes:

Monosync: 8-bit sync character matched by the Receiver

Bisync: Most significant (second) 8 bits of the 16-bit Transmit and Receive sync characters

SDLC: You must program the flag character, 011111110, into Control Register 7 for flag matching by the MPSC² Receiver.

3.9.11 Status Register 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Break/ Abort	Idle CRC	$\overline{CTS}$	Sync Status	$\overline{DCD}$	Transmitter Buffer Empty	Interrupt Pending	Received Character Available

Figure 3-10 Status Register 0

Received Character Available (D_0)

When this bit is set, it indicates that one or more characters are available in the receiver Buffer for the processor to read. Once all of the available characters have been read, the MPSC² resets this bit until a new character is received.

Interrupt pending (D_1 - Channel A Only)

The Interrupt Pending bit is used with the Interrupt Vector Register (Status Register 2) to make it easier to determine the MPSC²'s interrupt status, particlarly in Non-vecotred Interrupt Mode where the processor must poll each device to determine the interrupt source. In this mode, Interrupt Pending is set when you read Status Register 2B, the PRI input is active (low) and the MPSC² is requesting interrupt service.

You need not analyze the status registers of both channels to determine if an interrupt is pending. If Status Affects Vector is enabled and Interrupt Pending is set, the vector you read from SR2 contains valid condition information.

in Vectored Interrupt Mode, Interrupt Pending is set during the Interrupt Acknowledge Cycle (on the leading edge of the 2nd INTA pulse) when the MPSC² is the highest priority device requesting other pending interrupt requests, InterruptPending is reset when the End of Interrupt command is issued.

Transmitter Buffer Empty (D_2)

This bit is set whenever the Transmitter Buffer is empty, except during the transmission of CRC (the MPSC² uses the buffer to facilitate this function). After a reset, the buffer is considered empty and Transmit Buffer Empty is set.

External/Status Flags

The following status bits reflect the state of the various conditions that cause an External/Status interrupt. The MPSC² latches all External/Status bits whenever a change occurs that would cause an Extemal/Status interrupt ,regardless of whether this interrupt is enabled). This allows you to capture transient status changes on these lines with relaxed software timing requirements (see Appendix A for detailed timing specifications).

When you operate the MPSC² in interrupt-driven mode for External/Status Interrupts, you should read Status Register 0 when this interrupt occues and issue a Reset Extemal/Status Interrupt Command to teenable the interrupt and the latches. To poll these bits without interrupts, you can issue the Reset Extemal/Status Interrupt command to first update the status to reflect the current values.

DCD (D_3)

This bit reflects the inverted state of the DCD Input. When DCD is low, the DCD status bit is high. Any transition on this bit causes an External/Status Interrupt request.

Sync Status (D_4)

The meaning of this bit depends on the operating mode of the MPSC².

Asynchronous Mode: Sync Status reflects the inverted state of the SYNC Input. When SYNC is low, Sync Status is high. Any transition on this bit causes an External/Status Interrupt Request.

External Synchronization Mode: Sync status operates in the same manner as Asynchronous Mode. The MPSC²'s Receiver Suhynchronization logic is also tied to the Sync Status bit in External Synchronization Mode and a low-to-high transition (Sync Input going low) informs the receiver that synchronization has been achieved and character assembly begins (see Appendix A for detailed timing information).

A low-to-high transition on the SYNC Input indicates that synchronization has been lost and is reflected both in Sync Status becoming zero and the generatio of an External/Status Interrupt. The Receiver remains in the Receiver Data Phase until you set the Enter Hunt Phase bit in Control Register 3.

Monosync, Bisync SDLC Modes: In these modes, Sync status indicates whether the MPSC² Receiver is in the Sync Hunt or Receive Data Phase of operation. A 0 indicates that the MPSC² is in the Receive Data Phase and a 1 indicates that the MPSC² is in the Sync Hunt Phase, as after a reset or setting the Enter Sync Hunt Phase bit. As in the other modes, a transition on this bit causes an External/Status interrupt to be issued. You should note that Entering Sync Hunt Phase after either a Reset or when programmed causes an External/Status Interrupt request which you may clear immediately with a Reset External/Status Interrupt command.

$\overline{CTS}$ (D_5)

This bit reflects the inverted state of the CTS input. When CTS is low, the CTS status bit is high. Any trasition on this bit causes an External/Status Interrupt request.

Idle/CRC (D_6)

This bit indicates the state of the Idle/CRC latch used in Synchronous and SDLC modes. After Reset this bit is 1, indicating that when the Transmitter is completely empty, the MPSC² enters Idle Phase and automatically transmits sync or flag characters.

A zero indicates that the latch has been teset by the Reset Idle/CRC latch Command. When the Transmitter is completely empty, the MPSC² sends the 16-bit CRC character and sets the latch again. An External/Status interrupt is issued when the latch is set, indicating that CRC is being sent. No interrupt is issued when the latch is reset.

Break/Abort (D_7)

In Asynchronous mode, this bit indicates the detection of a Break Sequence (a null character plus framing error, that occurs when the RxD input is held low (spacing) for more than 1 character time)/ Break/ABort is reset when RxD returns high (marking).

In SDLC mode, Break/Abort indicates the detection of an Abort Sequence when 7 or more ones are received in sequence. It is reset when a zero is received.

Any transition of the Break/Abort bit causes an External/Status Interrupt.

3.9.12 Status Register 1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
End of SDLC Frame	CRC Framing Error	Overrun Error	Parity Error	SDLC Residue Code			All Sent

Figure 3-11 Status Register 1

All Sent (D_0)

In Asynchronous mode, this bit is set when the transmitter is empty and reset when a character is present in the transmitter buffer or Shift Register. This feature simplifies your modem control software routines. In Synchronous and SDLC modes, this bit is always set to 1.

SDLC Residue Code (D_1-D_3)

Since the data portion of an SDLC message can cnsist of any number of bits and not necassarily an integral number of characters, the MPSC^2 features special logic to determine and report when the End of Frame flag has been received, the boundary between the data field, and the CRC character in the last few data characters that were just read.

When the End of Frame condition is indicated, that is, Status Register 1 D_7 = 1 and Special/Receive Condition Interrupt (if enabled), the last bits of the CRC character are in the Receiver Buffer. The Residue Code for the frame is valid in the Status Register 1 byte associated with that data character (remember SR1 tracks the received data in its own buffer).

The mening of the Residue Code depends upon the number of bits/characters specified for the Receiver. The previous character refers to the last character read before the End of Frame, etc.

8 Bits/Character

D_3	D_2	D_1	Previous Character	2nd Previous Character
1	0	0	CCCCCCCC	CCCCCDDD
0	1	0	CCCCCCCC	CCCCDDDD
1	1	0	CCCCCCCC	CCCDDDDD
0	0	1	CCCCCCCC	CCDDDDDD
1	0	0	CCCCCCCC	CDDDDDDD
0	1	1	CCCCCCCC	DDDDDDDD (no residue)
1	1	1	CCCCCCCD	DDDDDDDD
0	0	0	CCCCCCDD	DDDDDDDD

7 Bits/Character

D_3	D_2	D_1	Previous Character	2nd Previous Character
1	0	0	CCCCCCC	CCCCCDD
0	1	0	CCCCCCC	CCCCDDD
1	1	0	CCCCCCC	CCCDDDD
0	0	1	CCCCCCC	CCDDDDD
1	0	0	CCCCCCC	CDDDDDD
0	1	1	CCCCCCC	DDDDDDD (no residue)
0	0	0	CCCCCCD	DDDDDDD

6 Bits/Character

D_3	D_2	D_1	Previous Character	2nd Previous Character
1	0	0	CCCCCC	CCCCCD
0	1	0	CCCCCC	CCCCDD
1	1	0	CCCCCC	CCCDDD
0	0	1	CCCCCC	CCDDDD
1	0	1	CCCCCC	CDDDDD
0	0	0	CCCCCC	DDDDDD (no residue)

5 Bits/Character

D_3	D_2	D_1	Previous Character	2nd Previous Character
1	0	0	CCCCC	DDDDD (no residue)
0	1	0	CCCCD	DDDDD
1	1	0	CCCDD	DDDDD
0	0	1	CCDDD	DDDDD
0	0	0	CDDDD	DDDDD

Table 3-12 Residue Codes

3.9.13 Special Receive Condition Flags

The status bits described below (Parity Error (If Parity Is a Special Receive condition is enabled), Receiver Overrun Error, CRC/Framing Error, and End of SDLC Frame), all represent Special Receive Conditions.

When any of these conditions occurs and interrupts are enabled, the MPSC^2 issues an interrupt request. In addition, if you enabled Condition Affects Vector mode, the vector generated (and the contents of SR2B for non-vectored interrupts) is different from that of a Received Character Available condition. Thus, you need not analyze SR1 with each character to determine that an error has occurred.

As a further convenience, the Parity Error and Receiver Overrun Error flags are latched, that is, once one of these errors occurs, the flag remains set for all subsequent characters until reset by the Error Reset command. With this facility, you need only read SR1 at the end of a message to determine if either of these errors occurred anywhere in the message. The other flags are not latched and follow each character available in the Receiver Buffer.

Parity Error (D_4)

This bit is set and latched when parity is enabled and the received parity bit does not match the sense (odd or even) caldulated from the data bits.

Receiver Overrun Error (D_5)

This error occurs and is latched when the receiver buffer already contains three characters and a fourth character is completely received, overwriting the last character in the buffer.

CRC/Framing Error (D_6)

In Asynchronous mode, a framing error is flagged (but not latchedd) when no stop bit is detected at the end of a character (i.e. RxD is low 1 bit time after the center of the last data or parity bit). When this condition occurs, the MPSC^2 waits an additional 1/2 bit time before sampling again so that the framing error is not interpreted as a new start bit.

In Synchronous and SDLC modes, this bit indicates the result of the comparison between the current CRC result and the appropriate check value and is usually set to 1 since a message rarely indicates a correct CRC result until correctly completed with the CRC check character. Note that a CRC error does not result in a Special Receive Condition Interrupt.

End of SDLC Frame (D_7)

This flag is used only in SDLC mode to indicate that the End of Frame flag has been received and that the CRC error flag and Residue Code is valid. You can reset this flag at any time by issuing an Error Reset command. The MPSC^2 also automatically resets this bit for you on the first character of the next message frame.

3.9.14 Status Register 2

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Interrupt Vector							

Figure 3-12 Status Register 2 (Channel B)

Interrupt Vector (D_0 - D_7) (Channel B Only)

Reading Status Register 2B returns the interrupt vector that you programmed into Control register 2B. If condition affects Vector Mode is enabled, the value of the Vector is modified as follows:

D_2	D_1	D_0	Condition
1	1	1	No Interrupt Pending
0	0	0	Channel B Transmitter Buffer Empty
0	0	1	Channel B External/Status Change
0	1	0	Channel B Special Receive Condition
1	0	0	Channel A Transmitter Buffer Empty
1	0	1	Channel A External/Status Change
1	1	0	Channel A Received Character Available
1	1	1	Channel A Special Receive Condition

Table 3-13 Condition Affects Vector Modifications

As you can see, code 111 can mean either Channel A Special Receive Condition of No Interrupt Pending. you can easily distinguish between the two by examining the Interrupt Pending bit (D_1) of Status Register 0, Channel A. remember, in non-vectored interrupt mose you must read the vector register first for Interrupt Pending to be valid.

3.10 MPSC^2 Programming Examples

3.10.1 Asynchronous Mode

```
Init:
        ISSUE Channel Reset Command (CR0)
        SET Bus Interface Options (CR2A)
        SET Interrupt Vector (CR2B) - if used
        SET Operating Mode (CR4):
                Asynchronous Mode
                Parity Select
                #of Stop Bits,
                Clock Rate.
        SET Receive Enable, Auto Enables, Receive Character Length (CR2)
        SET Transmit Enable, Modem Controls Transmit Char. Length (CR5)
        ISSUE Reset External/Status Interrupt command
        SET Transmit Interrupt Enable, Receive Interrupt On Every
                Character, External Interrupt Enable, Wait Mode Disable.
```

End of Initialization

```
Send:
        ISSUE First Byte To MPSC²
        RETURN To Main Program OR Wait

Interrupt:
        CASE Interrupt Type to DO:

Character Received:
        READ Character from MPSC²
        PROCESS Character
        ISSUE End Of Interrupt Command
        RETURN From Interrupt

Special Receive Condition:
        READ SR1
        ISSUE Error Reset Command
        CALL Special Error Routine
        ISSUE End Of Interrupt Command
        RETURN From Interrupt

Transmitter Buffer Empty:
        IF Last Character Transferred was End of Message
                THEN ISSUE Reset Transmit Interrupt/DMA Pending Command
        ELSE
                Transfer Next Character to MPSC²
        ISSUE End Of Interrupt Command
        RETURN From Interrupt

External/Status Change:
        READ SR1
        CALL Special Condition Routine
        ISSUE End Of Interrupt Command
        RETURN From Interrupt
```

End of Interrupt Case

```
Terminate Transmit:
        RESET Transmit Enable, RTS (CR5)
        RETURN

Terminate Receive:
        RESET Receive Enable (CR1)
        RESET DTR (CR5)
```

3.10.2 Synchronous Operation Example

```
Unit:
        ISSUE Channel Reset Command
        SET Interface option (CR2A)
        SET Interrupt vector (CR2B)
        SET Parity Mode, Sync Mode, 1x Clock (CR4)
        SET Sync Characer 1 (CR6)
        SET Sync Character 2 (CR7)
        RETURN

Initiate Transmit:
        ISSUE Reset external/status interrupt command
        SET External interrupt enable, Transmit interrupt enable
        Wait Enable, Wait on Transmit (CR1)
        SET Transmit Enable, ¢ of Bits/character, RTS, CRC Polynomial Select
```

Note: Transmitter is now enabled and will automatically begin sending Sync characters. WAIT several character times (a good idea to help system gain synchronization)

```
Next Message:
        ISSUE Reset transmit CRC command

Character:
        GET character
        IF character is to be included in CRC
        THEN
                SET CRC generator on (CR5)
        ELSE
                SET CRC generator off (CR5)
        WRITE character to MPSC2 (processor will "Wait" until Transmitter
                buffer is empty)
        IF character was not the last
        THEN
                GOTO send character (do next character)
        ELSE
                SET CRC generator on (CR5)
        ISSUE reset Idle/CRC latch command
        WAIT for external /status interrupt indicating CRC being sent
        IF next message is ready to be transmitted
        THEN
                GOTO next message (next message will be sent immediately
                following CRC)
        ELSE
                WAIT for transmit buffer empty interrupt indicating trailing
                        sync being sent
```

SET transmitter enable off, RTS off (CR5)

End of Transmit Routine

Synchronous Receive Routine

Receive Message:
SET external/status interrupt enable, receive interrupt on first character mode, wait enabled, wait on receive (CR1)
SET Receiver enable on
sync character load inhibit
#of bits/characters (CR1)
SET DTR on (CR5)
ISSUE reset external status interrupt command
ISSUE enable interrupt on next received character command
ISSUE error reset command

Receiver is now enabled and in the Hunt Phase

WAIT for external/status interrupt (indicating synchronization has been achieved)
ISSUE error reset command
WAIT for received character available interrupt (first synchronization has been achieved)
ISSUE reset CRC checker command
SET sync character load inhibit off

Get character:
GET character from MPSC[2] (processor will "WAIT" until at least 1 character is available)
IF character is to be included in CRC calculation
THEN
Turn CRC checker on (CR3)
ELSE
SET CRC checker off (CR3)
IF character is part of message data
THEN
SAVE character in memory
IF character was not end of message
THEN
GOTO read character

Note: End of Message

SET CRC checker on
READ 2 CRC characters
READ 2 characters (these characters may be part of the next message but must be read before CRC will be valid)
READ SR1 (this must be done immediately so that next character status will not overwrite)
IF parity or Overrun or CRC = Error

```
THEN
        GOTO error processor

IF more messages are to be received
THEN
        GOTO get next message
ELSE
        SET DTR off
SET receive enable off
SET external/status interrupts off,
        receiver interrupt mode disabled (CR1)
RETURN
```

3.11 Handling an SDLC Underrun Fault

Since SDLC-type protocols do not allow flags to be imbedded within a message as filler, a fault condition can sometimes occur where the Transmitter runs out of data to send. This situation is particularly common in interrupt-driven systems that are heavily task-loaded. You can use the MPSC²'s Idle/CRC Latch feature to detect thses underrun faults and abort the message before an erroneous End of Frame Flag is sent. This is accomplished by issuing a Reset Idle/CRC Latch command to the MPSC² immediately after loading it with the first character of the message. If an underrun condition occurs, the MPSC² automatically begins to send the CRC character calsulated up to that point and issues an External/Status Change Interrupt to indicate that the CRC is being sent. Since your software routine knows that the end of the message has not been reached, and underrun is indicated and your routine can immediately abort the message with a Send Abort Command.

3.12 Sending Synchronous Pad Characters

If you want to send one or more Pad characters between Synchronous Messages, you can do it two ways with the MPSC²:

1. When the MPSC² issues an External/Status Interrupt to indicate that CRC is being sent, you can begin loading your Pad Characters into the Transmitter.

2. Instead of loading Pad Characters in response to the above interrupt, you can simply change the programmed Sync Character on the fly, and the MPSC² will transmit Pads when it enters Idle Phase after sending CRC.

3.13 Transmitting Bisync Transparent Mode

Because of the ability to change the Sync Ragisters (CR6, CR7) on the fly, the MPSC² is truly compatible with Bisync Protocol's Transparent Mode. On entering this mode, program CR6 with the DLE character and, if an underrun condition occurs, the correct DLE-SYN sequence is transmitted. On leaving Transparent mode you should reset CR6 back to SYN.

Code	Function
000	Null
001	send SDLC Abort
010	Reset External/Status Report
011	Channel Reset
100	Enable Interrupt on Next character
101	Reset Reading Transmitter Interrupt/DMA Request
110	Error Reset
111	End of Interrupt (Channel A only)

Table 3-13 Commands

Code	Function
00	Null
01	Reset Receiver CRC Checker
10	Reset Transmitter CRC Generator
11	Reset IDLE/CRC Latch

Table 3-14 CRC Control Commands

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CR0	CRC Control Commmand		Command			Register Pointer		
CR1	Wait Function Enable	ø	Wait on Receiver Transmitter	Receiver Interrupt Mode		Condition Affects Vector	Transmitter Interrupt Enable	Ext/Status Interrupt Enable
CR2 (A)	Pin 10 SYNCB/ RTSB	ø	Interrupt Vector Mode			Priority	DMA Mode Select	
CR2 (B)	Interrupt Vector							
CR3	Number of Received Bits/Character		Auto Enables	Enter Hunt Phase	Receiver CRC Enable	Address Search Mode	Sync Character Load Inhibit	Receiver Enable
CR4	Clock Rate		Sync Mode		Number of Stop Bits Sync Mode		Parity Even/Odd	Parity Enable
CR5	DTR	Number of Transmitted Bits/Character		Send Break	Transmitter Enable	CRC Polynomial Select	$\overline{RTS}$	Transmitter CRC Enable
CR6	Sync Byte 1							
CR7	Sync Byte 2							

Figure 3-12 Control Registers

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SR0	Break/ Abort	Idle CRC	$\overline{CTS}$	Sync Status	$\overline{DCD}$	Transmitter Buffer Empty	Interrupt Pending	Received Character Available
SR1	End of SDLC Frame	CRC Framing Error	Overrun Error	Parity Error	SDLC Residue Code			All Sent
SR2 (B)	Interrupt Vector							

Figure 5XXXXX Status Registers

4. Parallel Ports

data on 6522 to follow

5. Audio System

Audio output from and (optionally) input to the system are provided by a built-in coder/decoder (CODEC), which uses a Continuously-Variable-Slope Delta modulation (CVSD) technique. This device produces audio output by converting a single bit, digital bit stream to an analog output.

The bit stream interface is provided by the 6852-SSDA chip which converts 8 bit data bytes from the processor to a bit-serial data stream for the CODEC. The SSDA also provides encode/decode control, via the DTR output, and a 3 byte FIFO buffer which reduces the real time processor servicing requirements.

Additional control of the audio section is provided by VIA 1 and VIA 3. The signals provided are Codec Clock and Volume Control. The encode/decode line, controlled by DTR from the SSDA, selects the desired audio function (input or output). Codec clock is a PB7 output (of VIA 3), a timer generated signal which determines Codec sampling rate (normally about 16KHz). Volume control, a CB2 output (of VIA 1), is a timer controlled recirculating shift register output and is an eight step, pulse-width-modulated ultra audio signal.

5.1 Input Signal Conditioning

The microphone amplifier utilizes half of an LM358 and a JFET in a variable gain amplifier used as a compressor. The attack time of the compressor is about 50 milliseconds. Release time is 250ms. Input signal amplitude range for acceptable record quality is about 5 to 75mv RMS. The second stage, 1/2 of a LM358, is a 3 pole butterworth low-pass filter with a cutoff frequency of about 3KHz. This filter eliminates "aliasing" in the CVSD modulator.

5.2 Output Conditioning and Power Amplifier

Following the CVSD, the output (playback) signal is low-pass filtered by another active, 3KHz cutoff butterworth filter (1/4 LM324). Following this stage, a CA4066B and its attendant drivers provide software controlled volume control by varying the duty factor of signal CODEC VOL. The frequency of this signal (including the produced sidebands) must be high enough to be above audible range; a minimum of 20 KHz is recommended. Playback power amplification is provided by an LM383. This stage also provides some roll-off to alleviate the above problem. The power stage will produce 4 watts of audio; thus, an external speaker should be used if above normal sound levels are programmed, since the internal speaker is rated at only 300 milliwatts.

5.3 SSDA Device Operation

5.3.1 SSDA Overview

At the bus interface, the SSDA appears as two addressable memory locations. Internally, there are seven registers: two read only and five write only registers. The read only registers are Status and Receive Data; the write only registers are Control 1, Control 2, Control 3, Sync Code and Transmit Data. The serial interface consists of serial input and output lines with independent clocks and four peripheral/modem control lines.

Data to be transmitted is transferred directly into the 3 byte Transmit Data First-In First-Out (FIFO) register from the data bus. Availability of the input to the FIFO is indicated by a bit in the Status register; once data is entered, it moves through the FIFO to the last empty location. Data at the output of the FIFO is automatically transferred from the FIFO to the Transmitter Shift register as the shift register becomes available to transmit the next character. If data is not available from the FIFO (underflow condition), the Transmitter Shift register is automatically loaded with either a sync code or an all 1's character. The transmit section should be programmed not to append parity onto the transmitted word.

For use in the VICTOR 9000 audio system, the SSDA should normally be programmed to use 8 bit, no parity, and External Sync mode. Then the DTR control selects the input or output function. However, for completeness and any special functions, all modes of SSDA operation are discussed in the following sections.

The method of serial data accumulating in the receiver depends on the synchronization mode selected. In External Sync mode, used for parallel/serial operation, the receiver is synchronized by the DCD (Data Carrier Detect) input and transfers successive bytes of data to the input of the Receiver FIFO. The Single-Sync-Character mode requires a match between the Sync-Code register and one incoming character before data transfer to the FIFO begins. In Two-Sync-Character mode, two sync codes must be received in sequence to establish synchronization. Subsequent to synchronization in any mode, data is accumulated in the shift register. Availability of a word at the FIFO output is indicated by a bit in the Status register.

The SSDA and its internal registers are selected by the address bus and the Read/Write (R/W) and Enable control lines. To configure the SSDA, Control registers are selected and the appropriate bits set. The Status register can be selected to read status.

The transmitter and receiver clock inputs are tied together. Signals to the microprocessor are the Data bus and Interrupt Request ($\overline{IRQ}$).

5.3.2 Initialization

During a power-up sequence, system reset sets the SSDA in an internally latched reset condition to prevent erroneous output transitions. The Sync-Code register, Control register 2, and Control register 3 should be loaded prior to the programmed release of the Transmitter and/or Receiver Reset bits. The bits in Control Register 1 should be cleared after the $\overline{RESET}$ line has gone high.

5.3.3 Transmitter Operation

Data is transferred to the transmitter section in parallel form via the data bus and the Transmit Data FIFO. The Transmit Data FIFO is a 3 byte register whose status is indicated by the Transmitter Data Register Available status bit (TDRA) and its associated interrupt enable bit. Data is transferred through the FIFO on negative edges of PHASE2 pulses. Two data transfer modes are provided in the SSDA: the 1 byte transfer mode provides for writing data to the transmitter section (and reading from the receiver section) one byte at a time; the 2 byte transfer mode provides for writing two data characters in succession.

Data automatically transfers from the last register location in the Transmit Data FIFO (when it contains data) to the Transmitter Shift register during the last half of the last bit of the previous character. A character is transferred into the Shift register by the Transmitter Clock. Data is transmitted LSB first.

When the Shift register becomes empty and data is not available for transfer from the Transmit Data FIFO, an underflow results, and a character is inserted into the transmitter data stream. This character will be either all 1s or the contents of the Sync-Code register, depending on the state of the Transmit Sync-Code-On-Underflow control bit.

Transmission is initiated by clearing the Transmitter Reset bit in Control register 1. When the Transmitter Reset bit is cleared, the first full positive half cycle of the Transmit Clock initiates the transmit cycle; the transmission of data (or underflow characters) begins on the negative edge of the Transmit Clock pulse which started the cycle. If the Transmit Data FIFO has not been loaded, an underflow character is transmitted. When the Transmitter Reset bit (Tx Rs) is set, the Transmit Data FIFO is cleared and the TDRA status bit is cleared. After one PHASE2 clock has occurred, the Transmit Data FIFO becomes available for new data and TDRA is inhibited.

5.3.4 Receiver operation

Data and a pre-synchronized clock are provided to the SSDA receiver section by means of the Receive Data (Rx Data) and Receive Clock (Rx Clk) inputs. The data is a continuous bit stream; character boundaries cannot be identified within the stream. The Receiver Shift register outputs are high when it is in the reset state.

5.3.5 Synchronization

The SSDA provides three operating modes related to character synchronization: One-Sync-Character mode, Two-Sync-Character mode, and External Sync mode. The External Sync mode requires synchronization and control of the receiving section through the Data Carrier Detect (DCD) input. The external synchronization source could consist of a direct control line from the transmitting end of the serial data link or from external logic designed to detect the start of a message block. The One-Sync-Character mode searches on a bit-by-bit basis until a match is achieved between the data in the Shift register and the Sync-Code register. A match indicates that character synchronization is complete and will be retained for the message block. In the Two-Sync-Character mode, the receiver searches for the first sync code match on a bit-by-bit basis and then looks for a second successive sync code character prior to establishing character synchronization. If the second sync code character is not received, the bit-by-bit search for the first sync-code resumes.

Sync codes received prior to the completion of synchronization (one or two character) are not transferred to the Receive Data FIFO. Redundant sync codes received during the preamble or sync codes which occur as fill characters can automatically be stripped from the data by setting the Strip-Sync control bit to minimize system loading. Character synchronization is retained until cleared by means of the Clear-Sync bit. This bit also inhibits the synchronization search routine.

5.3.6 Receiving data

Once synchronization has been achieved, subsequent characters are automatically transferred into the Receive Data FIFO and clocked through the FIFO to the last empty location by PHASE2 pulses. The Receiver Data Available status bit (RDA) indicates when data is available to be read from the last FIFO location (number 3) when in the 1 byte transfer mode. The 2 byte transfer mode causes the RDA status bit to indicate that data is available when the last two FIFO register locations are full. Available data in the Receive Data FIFO triggers an interrupt request if the Receiver Interrupt Enable bit (RIE) is set. The CPU should then read the SSDA Status Register, which indicates whether data is available for the CPU to read from the Receive Data FIFO register. The IRQ and RDA status bits are reset by a read from the FIFO. If more than one character has been received and is resident in the Receive Data FIFO, subsequent PHASE2 clocks cause the FIFO to update and the RDA and IRQ status bits to again be set. The read data operation for the 2 byte transfer mode requires a PHASE2 clock intervening between reads to allow the FIFO data to shift.

The other status bit which pertains to the receiver section is Receiver Overrun. The Overrun status bit is automatically set when a character is transferred to the Receive Data FIFO while the first register of the Receive Data FIFO is full. Overrun causes an interrupt if Error Interrupt Enable (EIE) has been set. The transfer of the overrunning character into the FIFO causes the previous character in the FIFO input register location to be lost. The Overrun status bit is cleared by reading the Status register (when the overrun condition is present) followed by a Receive Data FIFO register read. Overrun cannot occur and be cleared without providing an opportunity to detect its occurrence via the Status register.

5.4 SSDA Input/Output

5.4.1 SSDA Interface signals for CPU

The SSDA interfaces to the CPU with an 8 bit bi-directional data bus (ID0-ID7), a chip select line, a register select line, an interrupt request line, a read/write line, an enable line, and a reset line. These signals permit the CPU to have complete control over the SSDA.

SSDA Bidirectional Data (ID0-ID7)

The bidirectional data lines (D0-D7) allow for data transfer between the SSDA and the CPU. The data bus output drivers are three state devices that remain in the high-impedance (off) state except when the CPU performs an SSDA read operation.

SSDA Enable (PHASE2)

The Enable signal, PHASE2, is a high impedance TTL-compatible input that enables the bus input/output data buffers, clocks data to and from the SSDA, and moves data through the FIFO Registers. This signal is the continuous System PHASE2 1 Mhz clock.

Read/Write (R/W)

The Read/Write line is a high impedance input that is TTL compatible and is used to control the direction of data flow through the SSDA's input/output data bus interface. When Read/Write is high (CPU read cycle), SSDA output drivers are turned on if the chip is selected and a selected register is read. When it is low, the SSDA output drivers are turned off and the CPU writes into a selected register. The Read/Write signal is also used to select read-only or write-only registers within the SSDA.

Chip Select (CS)

This chip select line is a high impedance TTL compatible input line used to address the SSDA. The SSDA is selected when CS is low. Transfers of data to and from the SSDA are performed under the control of the Enable signal, Read/Write, and Register Select.

Register Select (RS)

The Register Select line is a high impedance input that is TTL compatible. A high level is used to select Control registers C2 and C3, the Sync Code register, and the Transmit/Receive Data registers. A low level selects the Control 1 and Status registers (see Table 5-1). This line is driven by the A0 bit of the system address bus.

Interrupt Request (IRQ)

Interrupt Request is a TTL compatible, open drain (no internal pullup), active-low output that is used to interrupt the CPU. The Interrupt Request remains low until cleared by the CPU.

$\overline{\text{RESET}}$

Input

The $\overline{\text{RESET}}$ input provides a means of resetting the SSDA from an external source. In the low state, the $\overline{\text{RESET}}$ input causes the following:

- The Receiver Reset (Rx Rs) and Transmitter Reset (Tx Rs) bits are set, causing both the receiver and transmitter sections to be held in a reset condition.
- Peripheral Control bits PC1 and PC2 are reset to zero, causing the SM/DTR output to be high.
- The Error Interrupt Enable (EIE) bit is reset.
- An internal synchronization mode is selected.
- The Transmitter Data Register Available (TDRA) status bit is cleared and inhibited.

When $\overline{\text{RESET}}$ returns high (the inactive state), the transmitter and receiver sections remain in the reset state until the Receiver Reset and Transmitter Reset bits are cleared via the bus under software control. The Control Register bits affected by $\overline{\text{RESET}}$ (Rx Rs, Tx Rs, PC1, PC2, EIE, and E/I Sync) cannot be changed when $\overline{\text{RESET}}$ is low.

5.4.2 Clock Inputs

Separate high impedance TTL compatible inputs are driven by a common source for clocking transmitted and received data. The source is the CB2 signal from the Control Port VIA.

Transmit Clock (Tx Clk)

The Transmit clock input is used to clock out of transmitted data. The transmitter shifts data on the negative transition of the clock.

Receive Clock (Rx Clk)

The Receive clock input is used to clock in received data. The clock and data must be synchronized externally. The receiver samples the data on the positive transition of the clock.

5.4.3 Serial Input/Output Lines

Receive Data (Rx Data)

The Receive Data line is a high impedance TTL compatible input through which data is received in a serial format. Data rates may be from 0 to 600 kbs.

Transmit Data (Tx Data)

The Transmit Data output line transfers serial data to a modem or other peripheral. Data rates may be from 0 to 600 kbs.

5.4.4 SSDA Registers

Seven registers in the SSDA can be accessed by means of the bus. The registers are defined as read-only or write-only according to the direction of information flow. The Register Select input (RS) selects two registers in each state, one being read-only and the other write-only. The Read/Write input (R/W) defines which pair is actually accessed. Four registers (two read-only and two write-only) can be addressed via the bus at any particular time. These registers and the required adressing are defined in Table 5-1.

REG	INP		CTL		REGISTER CONTENT							
	RS	RW	AC2	AC1	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Status (S)	0	1	X	X	Interrupt Request (IRQ)	Receiver Parity Error	Receiver Overrun (RX Ovrn)	Transmitter Underflow (TUF)	Clear to Send (CTS)	Data Carrier Detect (DCD)	Transmitter Data Register Available (TDRA)	Receiver Data Available (RDA)
Control (C1)	0	0	X	X	Address Control 2 (AC 2)	Address Control 1 (AC 1)	Receiver Interrupt Enable (RIE)	Transmitter Sync (TIE)	Clear Characters (Strip Sync)	Strip Sync Reset (Tx Rs)	Transmitter Reset (Rx Rs)	Receiver
Receive Data FIFO	1	1	X	X	D7	D6	D5	D4	D3	D2	D1	D0
Control 2 (C2)	1	0	0	0	Error Interrupt Enable (EIE)	Transmit Sync Code on U'flo (TX Sync)	Word Length Select 3 (WS 3)	Word Length Select 2 (WS2)	Word Length Select 1 (WS1)	1-Byte/2-Byte Transfer 1-/2-Byte	Peripheral Control 2 (PC2)	Peripheral Control 1 (PC1)
Control 3	1	0	0	1	Not Used	Not Used	Not Used	Not Used	Clear Transmitter Underflow Status (CTUF)	Clear CTS Status (Clear CTS)	One-Sync-Character/ Two-Sync-Character Mode Ctl	External/ Internal Sync Mode Control (E/I Sync)
Sync Code	1	0	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Transmit	1	0	1	1	D7	D6	D5	D4	D3	D2	D1	D0

X - Don't Care

Table 5-1

5.4.5 Control Register 1 (C1)

Control register 1 is an 8 bit write-only register that can be directly addressed from the data bus. Control register 1 is addressed when RS equals zero.

Receiver Reset (Rx Rs), C1 Bit 0

The Receiver Reset control bit provides both a reset and inhibit function to the receiver section. When Rx Rs is set, it clears the receiver control logic, sync logic, error logic, Rx Data FIFO Control, Parity Error status bit, and DCD interrupt. The Receiver Shift register is set to "ones". The Rx Rs bit must be cleared after the occurrence of a low level on $\overline{\text{RESET}}$ in order to enable the receiver section of the SSDA.

Transmitter Reset (Tx Rs), C1 Bit 1

The Transmitter Reset control bit provides both a reset and inhibit to the transmitter section. When Tx Rs is set, it clears the transmitter control section, Transmitter Shift register, Tx Data FIFO Control (the Tx Data FIFO can be reloaded after one PHASE2 clock pulse), the Transmitter Underflow status bit, and the CTS interrupt, and inhibits the TDRA status bit (in the one-sync-character and two-sync-character models). The Tx Rs bit must be cleared after the occurrence of a low level on $\overline{\text{RESET}}$ in order to enable the transmitter section of the SSDA. If the Tx FIFO is not preloaded, it must be loaded immediately after the Tx Rs release to prevent a transmitter underflow condition.

Strip Synchronization Characters (Strip-Sync), C1 Bit 2

If the Strip-Sync bit is set, the SSDA automatically strips all received characters which match the contents of the Sync-Code register. The characters used for synchronization (one or two characters of sync) are always stripped from the received data stream.

Clear Synchronization (Clear-Sync), C1 Bit 3

The Clear-Sync control bit provides the capability of dropping receiver character synchronization and inhibiting resynchronization. The Clear-Sync bit is set to clear and inhibit receiver synchronization in all modes and is reset to zero to enable resynchronization.

Transmitter Interrupt Enable (TIE), C1 Bit 4

TIE enables both the Interrupt Request output (IRQ) and Interrupt Request status bit to indicate a transmitter service request. When TIE is set and the TDRA status bit is high, the IRQ output goes low (the active state), and the IRQ status bit goes high.

Receiver Interrupt Enable (RIE), C1 Bit 5

RIE enables both the Interrupt Request output (IRQ) and the Interrupt Request status bit to indicate a receiver service request. When RIE is set and the RDA status bit is high, the IRQ output goes low (the active state), and the IRQ status bit goes high.

Address Control 1 (AC1) and Address Control 2 (AC2) C1 Bits 6 and 7

AC1 and AC2 select one of the write-only registers (Control 2, Control 3, Sync-Code, or Tx Data FIFO), as shown in Table 5-1, when RS equals one and R/W equals zero.

5.4.6 Control Register 2 (C2)

Control register 2 is an 8 bit write-only register which can be programmed from the bus when the Address Control bits in Control register 1 (AC1 and AC2) are reset and RS equals one and R/W equals zero.

Peripheral Control 1 and 2 (PC1 and PC2) C2 bits 0 and 1

The Peripheral Control 1 bit (PC1) and the Peripheral Control 2 bit (PC2) control the direction of data transfer and the selected CODEC function (Encode for receive; Decode for transmit). Control is accomplished by setting PC2 and setting PC1 to 00 for enabling the input (receive) function or to a 01 to enable the output (transmit) function. The DTR output is connected directly to the CTS input of the SSDA. Its complement is connected to the DCD input of the SSDA, as well as to the Encode/Decode select (pin 10) of the CODEC.

1 Byte/2 Byte Transfer (1 Byte/2 Byte), C2 Bit 2

When 1 Byte/2 Byte is set, the TDRA and RDA status bits indicate the availability of their respective data FIFO registers for a single byte data transfer. If 1 Byte/2 Byte is reset, the TDRA and RDA status bits indicate when two bytes of data can be moved without a second status read. An Enable pulse must occur between data transfers.

Word Length Selects (WS1, WS2, WS3), C2 Bits 3, 4, and 5

Word Length Select bits WS1, WS2, and WS3 select the word length (including parity) for the 7, 8, and 9 bits, as shown in Table 13.1.

Transmit Sync-Code on Underflow (Tx Sync), C2 Bit 6

When Tx Sync is set, the transmitter automatically sends a sync character when data is not available for transmission. If Tx Sync is reset, the transmitter transmits a Mark character (including the parity bit position) on underflow. If the Tx Sync bit is set when the underflow is detected, a pulse approximately the width of a Tx Clk high period occurs on the underflow output. Internal parity generation is inhibited during underflow except for sync code fill character transmission in 8 bit plus parity word lengths.

Error Interrupt Enable (EIE), C2 Bit 7

When EIE is set, the IRQ status bit goes high and the IRQ output goes low if:

- A receiver overrun occurs. The interrupt is cleared by reading the Status Register and reading the Rx Data FIFO.
- The transmitter has underflowed (in the Tx Sync On Underflow Mode). The interrupt is cleared by writing a one into the Clear Underflow, C3 bit 3, or Tx Reset.

When EIE is a 0, the IRQ status bit and the IRQ output are disabled for the preceding error conditions. A low level on the Reset input resets EIE to 0.

5.4.7 Control Register 3 (C3)

Control register 3 is a 4 bit write-only register that can be programmed from the bus when RS equals one and R/W equals zero and when Address Control bits AC1 equals one and AC2 equals zero.

External/Internal Sync Mode Control (E/I Sync), C3 Bit 0

When the E/I Sync Mode bit is high, the SSDA is in External Sync mode, and the receiver synchronization logic is disabled. Synchronization can be achieved by means of the DCD input. The DCD input is controlled directly by the DTR output, whose operation is described earlier in "Control Register 2, bits PCO and PC1." Both the transmitter and receiver sections operate as parallel to serial converters in External Sync mode. The Clear Sync bit in Control register 1 acts as a receiver sync inhibit when high to provide a bus controllable inhibit. The Sync-Code Register can serve as a transmitter fill character register and a receiver match register in this mode. A low on the $\overline{\text{RESET}}$ input resets the E/I Sync Mode bit, placing the SSDA in Internal Sync mode.

One-Sync/Two-Sync-Character Mode Control (1 Sync/2 Sync), C3 Bit 1

When the 1 Sync/2 Sync bit is set, the SSDA synchronizes on a single match between the received data and the contents of the Sync-Code register. When the 1 Sync/2 Sync bit is reset, two successive sync characters must be received prior to receiver synchronization. If the second sync character is not detected, the bit-by-bit search resumes from the first bit in the second character. Refer to the section of the Sync Code register for more detailed description.

Clear CTS Status (Clear CTS), C3 Bit 2

When a 1 is written into the Clear CTS bit, the stored status and interrupt are cleared. Subsequently, the CTS status bit reflects the state of the CTS input. The Clear CTS control bit does not affect the CTS input or its inhibit of the transmitter section. The Clear CTS command bit is self-clearing, so writing a 0 into this bit accomplishes nothing.

Clear Transmit Underflow status (CTUF), C3 Bit 3

When a 1 is written into the CTUF status bit, the CTUF bit and its associated interrupt are reset. The CTUF command bit is self-clearing.

5.4.8 Sync-code Register

The Sync-code register is an 8-bit register for storing the programmable sync code required for received data character synchronization in the One-Sync-Character and Two-Sync-Character modes. The Sync-code register also provides for stripping the sync/fill characters from the received data (a programmable option) and for automatic insertion of fill characters in the transmitted data stream. The Sync-code register is not used for receiver character synchronization in the External Sync mode; instead, it provides storage of receiver match and transmit fill characters.

The Sync-code register can be loaded when AC2 and AC1 are a 1 and a 0, respectively, and if R/W equals zero and RS equals one.

The Sync-code Register may be changed after the detection of a match with the received data (the first sync code having been detected) to synchronize with a double word sync pattern. (This sync code change must occur prior to the completion of the second character.) The sync match (SM) output can be used to interrupt the CPU system to indicate that the first eight bits have matched. The service routine would then change the Sync Code register to the second half of the pattern. Alternately, One-Sync-Character mode can be used for sync-codes of more than 8 bits by using software to check the second and subsequent bytes after reading them from the FIFO.

5.4.9 Parity for Sync Character

Transmitter

The Transmitter does not generate parity for the sync character except in 9 bit mode:

9 bit (8 bit + parity) generates an 8 bit sync character + parity

8 bit (7 bit + parity) generates an 8 bit sync character (no parity)

7 bit (6 bit + parity) generates a 7 bit sync character (no parity)

Receiver

The Receiver automatically strips the sync character(s) (there are two sync characters if 2 sync mode is selected) used to establish synchronization. Parity is not checked for these sync characters.

When the "strip-sync" bit is selected, the sync characters (fill characters) are stripped, and parity is not checked for the stripped sync (fill) characters. When the strip-sync bit is not selected (low), the sync character is assumed to be normal data and is transferred into FIFO after parity checking (if a parity format is selected).

STRIP SYNC (C1 BIT 2)	WSO-WS2 (DATA FORMAT C2 BIT 3-5)	OPERATION
1	X	No transfer of sync code and no parity check of sync code.
0	With parity	*Transfer data and sync codes. Parity check.
0	Without parity	*Transfer data and sync codes. No parity check.

* Subsequent to synchronization.

Table 5-2 Strip Sync Control Bit

Care should be exercised in selecting the sync character in the following situations:

- When Data format is (6 + parity) or (7 + parity)
- When Strip sync is not selected (low)
- When sync code is used as a fill character, and synchronization is established

The transmitter sends a sync character with parity, but the receiver checks the parity as if it were normal data. Therefore, the sync character should be chosen to match the parity check selected for the receiver in the special cases described in Table 5-2.

transfer mode. The Tx Data FIFO can be loaded with two bytes without an intervening status read. TDRA is inhibited by the Tx reset or reset. Upon Tx Reset, the Tx Data FIFO is cleared and then released on the PHASE2 clock pulse. The Tx Data FIFO can then be loaded with up to three data characters, even though TDRA is inhibited. This feature allows preloading data prior to the release of Tx Reset. A high level CTS input inhibits the TDRA status bit in either sync mode (One-Sync-Character mode or Two-Sync-Character mode). CTS does not affect TDRA in External Sync mode. Thus the SSDA is allowed to operate under the control of the CTS input with TDRA indicating the status of the Tx Data FIFO. The CTS input does not clear the Tx Data FIFO in any operating mode.

Data Carrier Detect (DCD), S Bit 2

A positive transition on the DCD input is stored in the SSDA until cleared by reading both Status and Rx Data FIFO. A one written into Rx Rs also clears the stored DCD status. The DCD status bit, when true, indicates that the DCD input has gone high. The reading of both Status and Receive Data FIFO allows Bit 2 of subsequent Status reads to indicate the state of the DCD input until the next positive transition.

Clear-to-Send (CTS), S Bit 3

A positive transition on the CTS input is stored in the SSDA until cleared by writing a 1 into the Clear CTS control bit or the Tx Rs bit. The CTS status bit, when true, indicates that the CTS input has gone high. The Clear CTS command (a 1 into C3 Bit 2) allows Bit 3 of subsequent Status reads to indicate the state of the CTS input until the next positive transition.

Transmitter Underflow (TUF), S Bit 4

When data is not available for the transmitter, an underflow occurs and is so indicated in the Status register (in the Tx Sync on underflow mode). The underflow status bit is cleared by writing a 1 into the Clear Underflow (CTUF) control bit or the Tx Rs bit. TUF indicates that a sync character will be transmitted as the next character. A TUF is indicated on the output only when the contents of the Sync Code Register is to be transferred (transmit sync code on underflow equals one).

Receiver Overrun (Rx Ovrn), S Bit 5

Overrun indicates that data has been received when the Rx Data FIFO is full, resulting in data loss. The Rx Ovrn status bit is set when Overrun occurs. The Tx Ovrn status bit is cleared by reading Status followed by reading the Rx Data FIFO or by setting the Rx Rs control bit.

Receiver Parity Error (PE), S Bit 6

The parity error status bit indicates that parity for the character in the last register of the Rx Data FIFO did not agree with selected parity. The parity error is cleared when the character to which it pertains is read from the Rx Data FIFO or when Rx Rs occurs. The DCD input does not clear the Parity Error or Rx Data FIFO status bits.

Interrupt Request (IRQ), S Bit 7

The Interrupt Request status bit indicates when the IRQ output is in the active state (IRQ output equals zero). The IRQ status bit is subject to the same interrupt enables (RIE, TIE, and EIE) as the IRQ output. The IRQ status bit simplifies status inquiries for polling systems by providing a single bit indication of service requests.

5.4.10 Receive Data First-in First-out Register (Rx Data FIFO)

The Receive Data FIFO register consists of three 8 bit registers and is used for buffer storage of received data. Each 8 bit register has an internal status bit that monitors its full or empty condition. Data is always transferred from a full register to an adjacent empty register. The transfer from register to register occurs on PHASE2 pulses. The RDA status bit is high when data is available in the last location of the Rx Data FIFO.

In an Overrun condition, the overrunning character is transferred into the full first stage of the FIFO register and causes the loss of that data character. Successive overruns continue to overwrite the first register of the FIFO. This destruction of data is indicated by the Overrun status bit. The Overrun bit is set when the overrun occurs and remains set until the Status Register is read and a read of the Rx Data FIFO occurs.

Unused data bits for short word lengths (including the parity bit) appear as zeros on the data bus when the Rx Data FIFO is read.

5.4.11 Transmit Data First-in First-out Register (TX DATA FIFO)

The Transmit Data FIFO register consists of three Shift registers used for buffer storage of data to be transmitted. Each 8 bit register has an internal status bit which monitors its full or empty condition. Data is always transferred from a full register to an adjacent empty register. The transfer is clocked by pulses.

The TDRA status bit is high if the Tx Data FIFO is available for data.

Unused data bits for short word lengths are handled as "don't cares." The parity bit is not transferred over the data bus since the SSDA generates parity at transmission.

When an Underflow occurs, the Underflow character is either the contents of the sync code register or an all ones character. The Underflow is stored in the Status register until cleared and appears on the Underflow output as a pulse approximately the width of a Tx Clk high period.

5.4.12 Status Register

The Status register is an 8 bit read-only register. It provides the real time status of the SSDA and the associated serial data channel. Reading the Status register is nondestructive. The method of clearing status bits depends upon the function each bit represents and is treated separately for each bit in the register, as described in the following sections.

Receiver Data Available (RDA), S Bit 0

The Receiver Data Available status bit indicates when receiver data can be read from the Rx Data FIFO. The presence of Receiver data is in the last register (#3) of the FIFO causes RDA bit to be high for the 1 byte transfer mode. In the 2-byte transfer mode, a high RDA bit indicates that the last two registers (#2 and #3) are full. The second character can be read without a second status read (to determine whether the character is available). Status must be read on a byte-by-byte basis if receiver data error checking is desired. The RDA status bit is reset automatically when data is not available.

Transmitter Data Register Available (TDRA), S Bit 1

The TDRA status bit indicates that data can be loaded into the Tx Data FIFO register. An empty first register (#1) of the Tx Data FIFO is indicated by a high level TDRA status bit in the 1-byte transfer mode. The first two registers (#1 and #2) must be empty for TDRA to be high when in the 2 byte

5.4.13 STATUS REGISTER

IRQ	Bit 7 - The IRQ flag is cleared when the source of the IRQ is cleared. The source is determined by the enables in the Control registers. TIE, RIE, EIE.
	Bits 6 to 0 - Indicate the SSDA status at a point in time, and can be reset as follows:
PE	Bit 6 - Read Rx Data FIFO, or a 1 into Rx Rs (C1 Bit 0).
Rx Ovrn	Bit 5 - Read Status and then Rx Data FIFO or a 1 into Rx Rs (C1 Bit-0).
TUF	Bit 4 - A 1 into CTUF (C3 Bit-3) or into Tx Rs (C1 Bit-1).
CTS	Bit 3 - A 1 into Clear CTS (C3 Bit-2) or a 1 into Tx Rs (C1 Bit-1).
DCD	Bit 2 - Read Status and then Rx Data FIFO or a 1 into Rx Rs (C1 Bit-0).
TDRA	Bit 1 - Write into Tx Data FIFO.
RDA	Bit 0 - Read Rx Data in FIFO.

5.4.14 Control Register 1

AC2,AC1	Bits 7,6 - Used to access other registers, as shown above.
RIE	Bit 5 - When 1, enables interrupt on RDA (S Bit-0).
TIE	Bit 4 - When 1, enables interrupt on TDRA (S Bit-1).
Clear Sync	Bit 3 - When 1, clears receiver character synchronization.
Strip Sync	Bit 2 - When 1, strips all sync codes from the received data stream.
Tx Rs	Bit 1 - When 1, resets and inhibits the transmitter section.
Rx Rs	Bit 0 - When 1, resets and inhibits the receiver section.

5.4.15 Control Register 3

CTUF	Bit 3 - When 1, clears TUF (S Bit 4), and IRQ if enabled.
Clear CTS	Bit 2 - When 1, clears CTS (S Bit 3), and IRQ if enabled.
1 Sync/2 Sync	Bit 1 - When 1, selects the one sync character mode; when 0, selects the two sync character mode.
E/I Sync	Bit 0 - When 1, selects the external sync mode; when 0, selects the internal sync mode.

5.4.16 Control Register 2

EIE — Bit 7 - When 1, enables the PE, Rx Ovrn, TUF, CTS, and DCD interrupt flags (S Bits 6 through 2).

Tx Sync — Bit 6 - When 1, allows sync code contents to be transferred on underflow, and enables the TUF Status bit and output. When 0, an all mark character is transmitted on underflow.

WS3,2,1 — Bits 5,4,3 As given in Table 5-1

BIT 4 WS3 LENGTH	BIT BIT WS2	5 3 WS1	BIT WORD
0	0	0	6 bits + even parity
0	0	1	6 bits + odd parity
0	1	0	7 bits, no parity
0*	1	1	8 bits, no parity
1	0	0	7 bits + even parity
1	0	1	7 bits + odd parity
1	1	0	8 bits + even parity
1	1	1	8 bits + odd parity

* This is the mode which should always be used.

Table 5-3 Word Length Selection

1 Byte/2 Byte — Bit 2 - When 1, enables the TDRA and RDA bits to indicate when a 1 byte transfer can occur. When 0, the TDRA and RDA bits indicate when a 2 byte transfer can occur.

PC2, PC1 — Bits 1 and 0 - As given in Table 5-4.

BIT 1 PC2S	BIT 0 PC1	SM/DTR OUTPUT AT PIN 5
0 Select audio output	0	1
1 Select audio input	0	0

Table 5-4 SM/DTR Output Control

5.5 CODEC Device Operation

The Continuously-Variable-Slope-Delta modulator (CVSD) is a simple alternative to more complex conventional conversion techniques in systems requiring digital communication of analog signals. The human voice is analog, but digital transmission of any signal over great distance is attractive. Signal/noise ratios do not vary with distance in digital transmission, and multiplexing, switching, and repeating hardware is more economical and easier to design. However, instrumentation Analog-to-Digital converters do not meet the communications requirements. The CVSD Analog-to-Digital is well suited to the requirements of digital communications and is an economically efficient means of digitizing voice inputs for transmission.

5.5.1 Delta Modulator

The innermost control loop of a CVSD converter is a simple delta modulator. A delta modulator consists of a comparator in the forward path and an integrator in the feedback path of a simple control loop. The inputs to the comparator are the analog input signal and the integrator output. The comparator output reflects the sign of the difference between the input voltage and the integrator output. That sign bit is the digital output and also controls the direction of ramp in the integrator. The comparator is normally clocked, producing synchronous and band-limited digital bit stream.

If the clocked serial bit stream is transmitted, received, and delivered to a similar integrator at a remote point, the remote integrator output is a copy of the transmitting control loop integrator output. To the extent that the transmitting integrator tracks the input signal, the remote receiver reproduces that input signal. Low pass filtering at the receiver output eliminates most of the quantizing noise if the clock rate of the bit stream is an octave or more above the upper band limit of the input signal. Input bandwidth cuts off above 3KHz, so clock rates from 8KHz up are possible. Thus, the delta modulator digitizes and transmits the analog input to a remote receiver. The serial, unframed nature of the data is ideal for communications networks. With no input at the transmitter, a continuous one/zero alternation is transmitted. If the two integrators are made leaky, then, during any loss of contact, the receiver output decays to zero and receive restart begins without framing when the receiver re-acquires. Similarly, a delta modulator is tolerant of sporadic bit errors.

5.5.2 Companding Algorithm

The fundamental advantages of the delta modulator are its simplicity and the serial format of its output. Its limitations are those caused by a limited digital bit rate. The analog input must be band limited and amplitude limited. The frequency limitations are governed by the Nyquist information rate relationships, and the amplitude capabilities are set by the gain and dynamic range of the integrators.

The frequency limits are bounded on the upper end; that is, for any input bandwidth there exists a clock frequency larger than that bandwidth transmits the signal with a specific noise level. However, the amplitude limits are bounded on both upper and lower ends. For any given signal level, one specific gain achieves an optimum noise level. Unfortunately, the basic delta modulator has a small dynamic range over which the noise level is constant.

The continuously-variable-slope circuitry provides increased dynamic range by adjusting the gain of the integrator. For a given clock frequency and input bandwidth, the additional circuitry increases the delta modulator's dynamic range. External to the basic delta modulator is an algorithm which monitors the past few outputs of the delta modulator in a simple shift register. The register is 2 bits long. The accepted CVSD algorithm simply monitors the contents of the shift register and indicates if it contains all ones or zeros. This condition is called coincidence. When it occurs, it indicates that the gain of the integrator is too small. The coincidence output charges a single pole low pass

filter. The voltage output of this "syllabic filter" controls the integrator gain through a pulse amplitude modulator whose other input is the sign bit or up/down control.

The simplicity of the all ones/all zeros algorithm should not be taken lightly. Many other control algorithms using shift registers have been tried. The key to the accepted algorithm is that it provides a measure of the average power or level of the input signal. Other techniques provide more instantaneous information about the shape of the input curve. The purpose of the algorithm is to control the gain of the integrator and to increase the dynamic range. Thus, a measure of the average input level is what is needed.

The algorithm is repeated in the receiver, and thus the level data is recovered in the receiver. Because the algorithm only operates on the past serial data, it changes the nature of the bit stream without changing the channel bit rate.

The effect of the algorithm is to compand the input signal. If the bit stream from a CVSD encoder is played into a basic delta modulator, the output of the delta modulator reflects the shape of the input signal, but all of the output will be at an equal level. Thus, the algorithm is needed at the output to restore the level variations. The bit stream on the channel behaves as if it came from a standard delta modulator with a constant level input.

The delta modulator encoder with the CVSD algorithm provides an efficient method for digitizing voice signals in a manner which is especially convenient for digital communications requirements.

PIN	FUNCTION
1	VDD (+5 volts)
2	Audio Ground. Connection to D/A ladders and comparator.
3	Audio Out Recovered audio out. Presents approximately 100Kohm source. Zero signal reference is VDD/2.
4	AGC (not used). A logic "low" level appears at this output when the recovered signal excursion reaches one-half of full scale value.
5	Audio Input. Externally AC coupled.
6	N/C
7	N/C
8	Ground Logic Ground
9	Clock Input
10	Encode/Decode. A low level selects the encode mode, a high level, the decode mode.
11	Alternate Plain Text (not used). A low level at this input causes a quieting pattern to be transmitted without affecting the internal operation of the CVSD.
12	Digital Data Input
13	Force Zero (not used). A low level at this input forces the transmitted output, the internal logic, and the recovered audio output of the CVSD into the "quieting" condition.
14	Digital Data Output

Table 5-5 Definitions and Functions of Pins

6. Keyboard

6.1 General

This chapter contains the specifications for the VICTOR 9000 keyboard plus information on the protocol used by the CPU to communicate with it.

6.2 Keyboard Layout

Key layouts vary from model to model in relation to the targeted application. The layout is broken into typewriter keys, command keys, and calculator keys. The typewriter pad has 58 possible key positions. The whole keyboard has a total of 104 possible key positions. The typewriter pad is sculptured; other pads are sloped. The layout uses one common PC Board, while the actual number of key positions occupied varies from model to model.

6.3 Protocol Definitions

The communication between the terminal processor and the keyboard is serial. The transmission is in 9 bit words. The first eight bits are the data byte, transmitted LSB first. The last bit is a stop bit.

The keyboard will return key numbers and key status through the eight data bits. The MSB of the key number returned by the keyboard is status which flags a key close or key open. An MSB of one indicates a key close, and an MSB of zero indicates a key open. The least significant 7 bits are the key number.

The stop bit is a zero from $\overline{\text{KBRDY}}$ low to $\overline{\text{KBACK}}$ low. The stop bit goes high before $\overline{\text{KBRDY}}$ goes high and remains high until the next transfer.

The keyboard indicates it has an event in its buffer with the $\overline{\text{KBRDY}}$ line. If transmission is idle, the keyboard can signal an event by taking the $\overline{\text{KBRDY}}$ line low. The high to low transition of $\overline{\text{KBRDY}}$ should flag an interrupt in the terminal processor. The keyboard should raise the $\overline{\text{KBRDY}}$ line on the negative transition of the $\overline{\text{KBACK}}$ line. Each event in the keyboard buffer will cause a transition of the $\overline{\text{KBRDY}}$ line. The keyboard transmission becomes idle after the positive edge of the $\overline{\text{KBACK}}$ line following the stop bit.

The keyboard times out the processor response to $\overline{\text{KBRDY}}$ low for 250 milliseconds. If the processor does not respond with a negative transition of $\overline{\text{KBACK}}$ clock within this time, the keyboard will drive $\overline{\text{KBRDY}}$ high and then restart the current transmission. This will allow the terminal processor to resynchronize to the keyboard data stream.

6.4 Keyboard Timing Diagram

Figure 6-1 illustrates keyboard timing

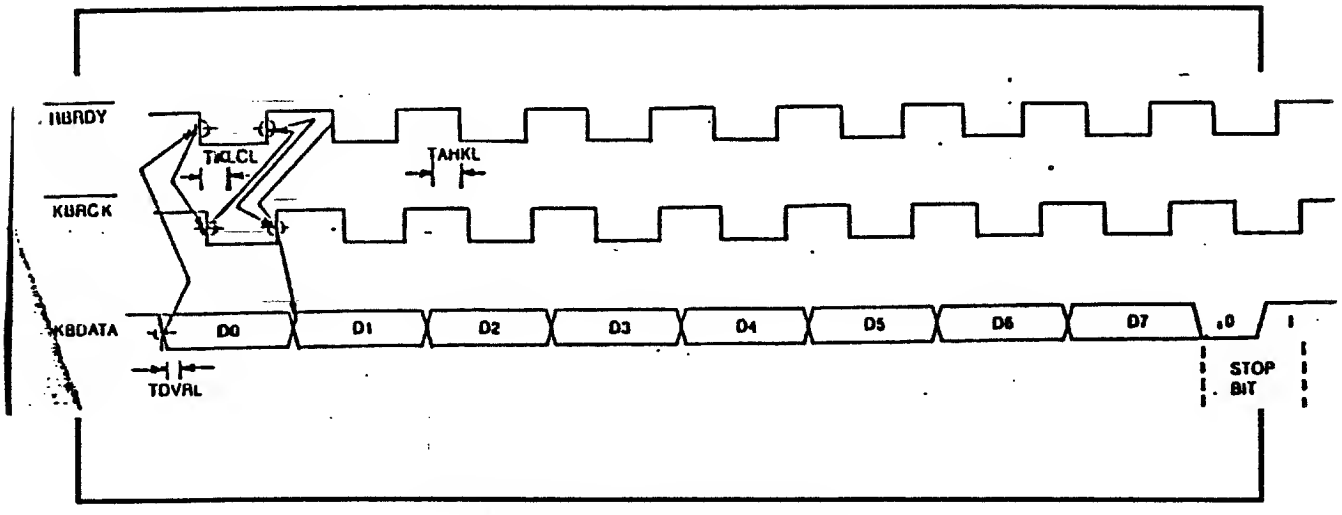

Figure 6-1 Keyboard Timing Diagram

6.5 Keyboard Connector Pin Assignments

The keyboard is connected to the CPU with a six conductor cable with the following signal assignments:

PIN(S)	NAME	FUNCTION
1,7	+5V	+5 volts at 250 ma
2,3	GROUND	System Ground
4	$\overline{\text{KBACK}}$	TTL Input. Driven by terminal processor. Transitions indicate acknowledgement of $\overline{\text{KBRDY}}$ transitions.
5	$\overline{\text{KBRDY}}$	TTL Output. Driven low by the keyboard to initiate handshake of each data bit of a transmission. Driven high after receipt of the negative edge of the $\overline{\text{KBACK}}$ line.
6	KBDATA	TTL Output. Changed after the positive edge of the $\overline{\text{KBACK}}$ line. Data must change no later than the negative edge of $\overline{\text{KBRDY}}$. The exception to this is the stop bit. Transfer of the stop bit is as follows: 1) Data line driven low at or before negative edge of $\overline{\text{KBRDY}}$. 2) Data line and $\overline{\text{KRBDY}}$ driven high following the negative edge of $\overline{\text{KBACK}}$. 3) Keyboard enters the Idle state after the positive edge of $\overline{\text{KBACK}}$.

Table 6-1 Pin Assignment

Key Travel	.150" – .200" (3.8mm – 5mm
Key Pretravel **(when applicable)**	.100" minimum (2.5 mm)
Actuation Force (Standard Key)	1.5 – 2.5oz (42.5 – 70 grams)
Actuation Force (Special Key)	3 – 5 oz (85 – 142 grams)
Reliability	>100 million cycles
Key Spacing	.70" – .80" (18 - 20 mm)
Key Sideplay	.018" .5mm
Key Rotation	<2°
Key Top Dimensions	.47" – .60" 12 – 15mm
Key Surface	Concave, textured (matte) unless position marked otherwise, low reflection, low glare.
Key Top Pull Resistance	Keytop shall be capable of withstanding 3 lbs (1.4 kg) pull without coming loose and 11 lbs (5 kg) in the direction of actuation without any damage to the key switch.

Table 6-5 Mechanical Specifications

PARAMETER	FUNCTION DESCRIPTION	REQUIRED TIMING MAX	MIN
TDVRL	KB data valid to $\overline{\text{KBRDY}}$ low	—	0
TRLCL	$\overline{\text{KBRDY}}$ low to $\overline{\text{KBACK}}$ low	250ms	—
TAHKL	$\overline{\text{KBACK}}$ high to $\overline{\text{KBRDY}}$ low (except after stop bit)	1ms	0

Table 6-2 Switching Characteristics

CODE	FUNCTION	DESCRIPTION
FE_{16}	Overflow	Key queue overflow. Keys have been lost.
FF_{16}	Dead	Keyboard dead or not connected.

Table 6-3 Reserved Keyboard Codes

6.6 Specifications

Input Voltage	+5V DC ±5%
Input Current	250 ma
Rollover	N Key
Connector Type	AMP 87551-7 or equivalent
Connector Spacing	0.1", 7 pin header

Table 6-4 Electrical Specifications

Operating Temperature	0°C – 50°C
Storage Temperature	−40°C – +60°C
Humidity	0 – 95% noncondensing
Flammability of Material	Self-extinguishing
Approvals	UL and VDE
Vibration	To be Specified
Shock (Operating)	10G peak (1/2 sinusoid) 10ms duration
Shock (Non-operating)	100G peak (1/2 sinusoid) 10ms duration

Table 6-6 Environmental Specifications

7. Disk Drive Assembly

As shown in Figure 7-1, the disk drive assembly is comprised of two floppy disk drive mechanisms, a disk drive interface board, and a chassis which also contains a speaker. The disk drive assembly provides the system with a minimum of 1.2 million bytes (formatted) of auxillary storage.

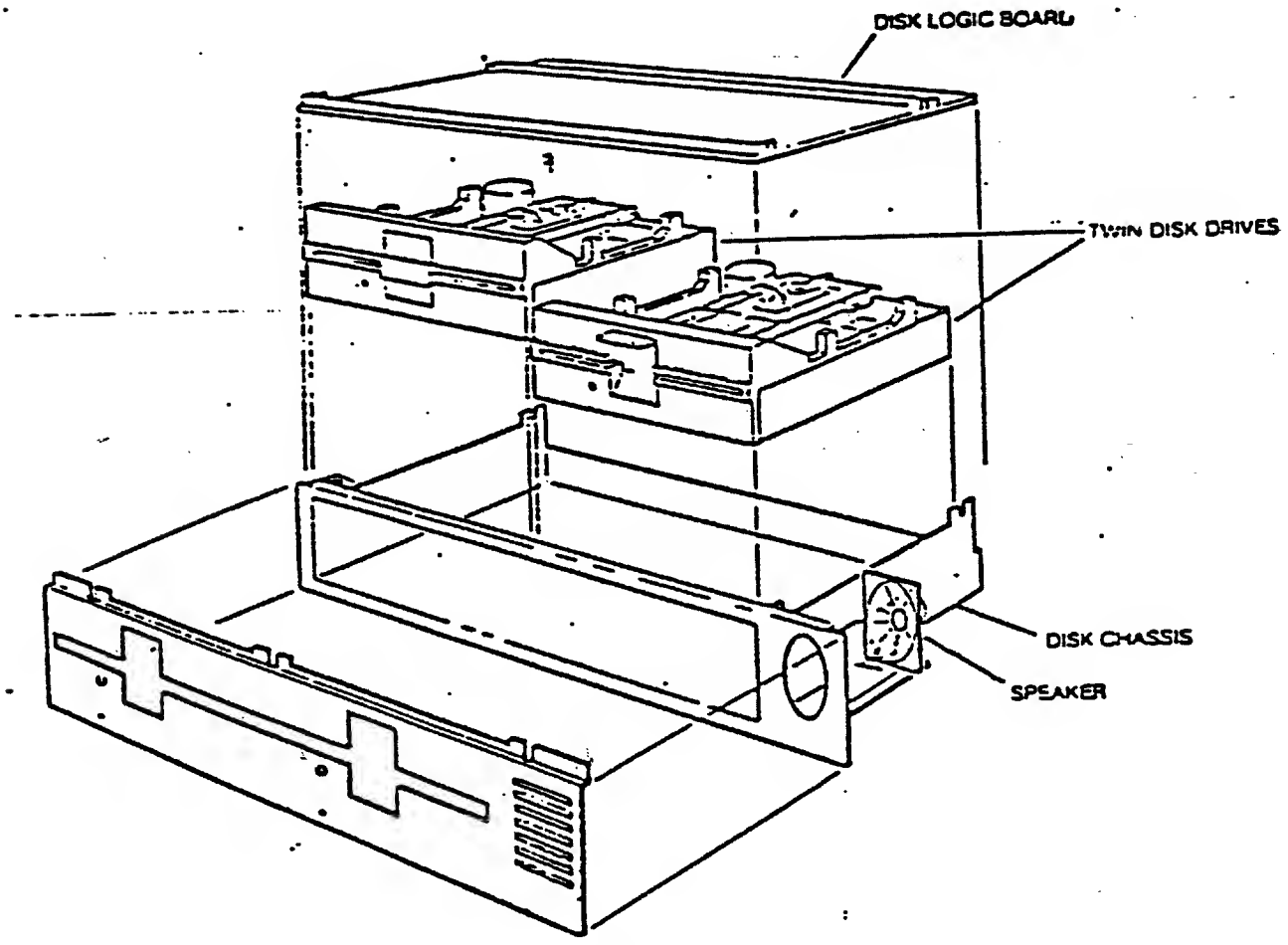

Figure 7-1. Disk Drive Assembly

The standard drive units are 5-1/4 inch, 80-track mechanisms, which operate with single-sided media. Track density is 96 tracks per inch, and recording density is maintained at approximately 8000 bits per inch on all tracks.

7.1 Functional Description of Disk Drive

The disk drive interface board provides all the low level operations required to convert binary information for storage on and retrieval from diskette. Status and drive control interface to the drives is also provided on the disk drive interface board.

The processing unit maintains functional control of the disk drive assembly.

7.2 Reading Data

The 8088 CPU transfers data from the disk to memory as byte-by-byte read operations. Before the data is transferred, the drive motor for the drive containing the disk is started, and the head is positioned to the correct track. The GCR read circuit provides sync detection and separation. (Sync is a special GCR pattern that does not occur in normal data fields. The sync pattern consists of 10 ones during a byte time; other GCR patterns cannot contain more than 8 ones during a byte time.) When the GCR read circuit detects a sync mark, it starts a counter that causes an interrupt to be sent to the CPU, if sync remains present for 6 byte times. This interrupt to the CPU, which is called SYN and is on the highest level interrupt input line to the interrupt controller, informs the CPU that a header sync mark has been detected.

7.3 Header Search

When a sync interrupt occurs while the CPU is searching for a sector, the CPU enters the controller software that will compare the sector header information with the sector requested (the sector header contains the data block ID, track numbers, the sector number, and the checksum). This compare function is performed by the CPU on a byte-by-byte basis. The GCR read circuit provides a data byte every 21.3 microseconds. In order to be able to keep up with the high data rate, the CPU uses a special instruction (WAIT) that stops processing until a byte- ready strobe occurs on the test input. The CPU then continues processing by reading the latched data byte and comparing it with the requested sector information.

If the sector is not the correct sector, the CPU returns from the interrupt and continues processing until the next header sync interrupt. Once the desired sector header has been found, the data transfer can begin.

7.4 Data Transfer

Before the CPU can read the data block of a sector, the clock recovery circuitry must be resynchronized. This is required because the data block is updated and can be written at any random phase relative to the header information. The data block sync mark is only 5 bytes long and is not detected by the header sync mark detection circuit (header sync marks must be at least 6 bytes in length). The CPU polls the sync input line until the data block sync is detected and then verifies that the byte following sync—the data block IO byte—is correct. If it is not correct, a "not data block IO error" is generated, and no data is transferred. Using the WAIT instruction, the CPU then transfers the following 512 bytes of sector information to the preset destination in memory. As the CPU moves the data to memory, it also computes the checksum. This resulting checksum is then compared with the checksum recorded in the data block.

If the check sums match, the data transfer is correct. Otherwise, error recovery by the CPU is needed.

7.5 Writing Data

Data transfer from memory to disk is performed by the CPU in much the same manner as for read operations. The disk drive motor is started and set to the proper speed, and the head is positioned at the correct track by the controller software. The CPU does a header search using the method described earlier in "Reading Data." When the desired header is matched, the CPU starts an update operation of the data portion of the sector and, before turning on the write current, times the GAP1 area. The 5 byte data block sync area is written. Next the 10 byte data block, and then 512 bytes of sector data are written from the preset location in memory. As the data is written, the CPU also creates the 2-byte checksum, which is written at the end of the data section.

The CPU also controls the trim erase timing of the read/write head. The purpose of trim erase is to erase any remaining portion of the old data section that was recorded from the sides of the new data section. At the end of the update, the write current is turned off, and, about 31 byte times later, the trim erase is turned off.

7.6 Verification

In order to ensure reliable data storage, all sector updates are followed by a verify operation. A verify operation is similar to a read operation, except that the data in memory is compared to the read disk data being transferred to memory. If any of the bytes do not compare correctly with the data in memory, an error is flagged, and an error recovery is performed by the CPU.

7.7 Formatting

A blank or new diskette must be formatted before it can be used. (Some programs, such as DCOPY, perform the formatting function implicitly.) Formatting is done by writing control information and dummy data blocks to all 80 tracks on the disk (see the "Track Format" and "Sector Format" sections under "Physical Description"). The format is a variable number of sectors per zone in soft sectored format. In order to achieve maximum speed tolerance on each diskette, the CPU performs an adaptive format procedure. Diskette speed variation (from unit to unit) causes the number of bytes on a track to vary. During format this problem is solved by always providing a fixed number of unused bytes to allow for the worst case speed. Instead of allowing the unused bytes to be wasted, the format procedure measures the size of the first track in each zone and then adjusts the gap to the size of the sector format.

This causes the physical sector size to remain constant regardless of diskette speed during format. This method allows the maximum possible tolerance to speed variation without requiring a gap at the end of the track to allow for speed variation. The Victor technique makes better use of the unused space by distributing it and using the additional intersector time to achieve stabilization of the clock recovery circuitry.

Refer to "Speed Control" and "Motor Speed Control" for more details on speed control.

7.8 Positioning

The head positioning mechanism for each drive is a four-phase stepper motor. The disk drive interface has drivers for each stepper motor which are controlled directly by the CPU. By properly sequencing the four phases of the stepper motor, the CPU can move the head of each drive in or out. All timing and control is done in software by the CPU. To reduce power consumption, the stepper motors are energized only when the drive is active; otherwise they are turned off by the CPU. The independent stepper drivers allow the CPU to perform overlapping seeks, resulting in higher system performance.

7.9 Speed Control

In order to attain maximum data capacity, the media passes under the head at a constant linear velocity. To attain this, the rotational period is varied as the radius of the track changes. The disk rotational speed is selected by the CPU. The actual speed control is performed by a single chip computer on the disk drive interface board. The CPU communicates with the speed control processor (SCP) by an eight bit port. On system power- up, the SCP uses a default speed table that allows the system to boot. Once the operating system software is loaded, the CPU writes a new speed table to the SCP that allows it to operate with the current 512-byte sectors. The SCP can be programmed with up to 15 different speeds.

7.10 Physical Description

The disk interface board contains the circuitry necessary to control both of the integrated system disk drives. This circuitry consists of drive motor speed control, read/write head positioning, data decoding and encoding, read channel electronics, and write channel electronics. The interface board receives functional control from the processor unit through a dedicated I/O bus.

7.11 Motor Speed Control

The traditional approach to storing data on floppy disks is to write data (using some encoding scheme) at a fixed rate, while rotating the disk at a constant speed. This results in several undesirable characteristics. Three major undesirable characteristics that were addressed by Victor are wasted capacity, large variation in the read signal amplitude, and low system tolerance to motor speed variation.

Since the circumference of the outermost track on the floppy is larger than the circumference of the innermost track (and, in fact, larger than all other tracks) the recording density on the outermost track is lower than on the innermost. The major limiting factor in recording on magnetic media is bit density (actually, flux reversal density), which means that the outer tracks contain less data than the inner tracks, unless adjustment is made to accommodate this problem.

Also, when the disk is rotated at a constant speed or RPM, the linear velocity of the head relative to the media varies from track to track. Since the amplitude of the recorded signal is partly a function of speed, the signal amplitude varies greatly from the outermost track (where it is highest) to the innermost track. This results in a read channel that has a lower signal- to-noise ratio than would be obtainable if all tracks were recorded with a constant amplitude signal.

The VICTOR 9000 overcomes these two problems by setting disk rotation speed according to the track circumference. This is done in a way that maintains a nearly constant bit density and a nearly constant linear velocity, hence a constant amplitude signal.

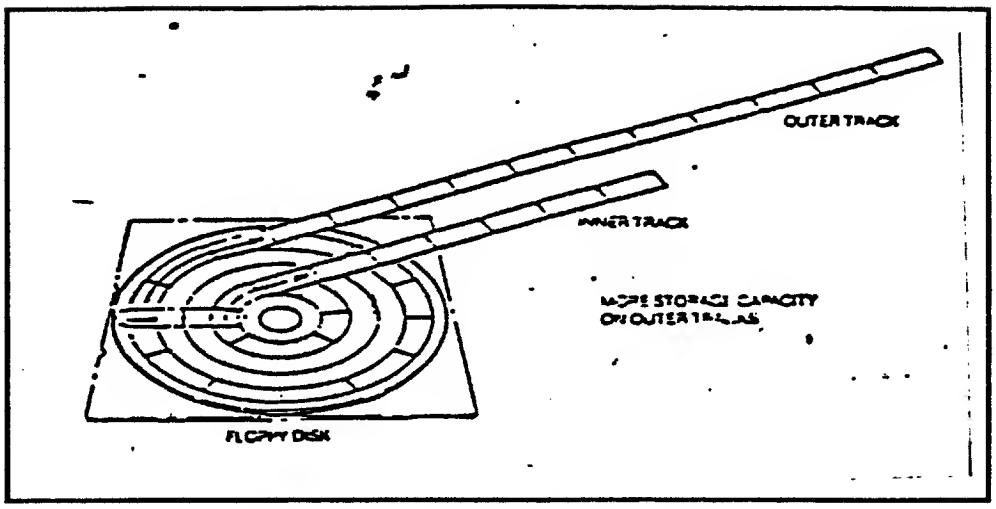

Figure 7-2. Disk Track and Sector Layout

Data written to the disk is organized into groups of 512 bytes (plus a number of synchronization and control information bytes). These groups are called sectors. Although the circumference of each track differs slightly, it is not possible to take advantage of the potential difference in capacity without using sectors of varying size. Therefore, the speed is changed only when this results in enough additional capacity for an extra sector. The disk is thus divided into groups of tracks, called zones. Each zone, when being read or written, causes the disk to rotate at a slightly different speed.

The third problem—low system tolerance to motor speed variations—is caused by a phenomena called bit shift or pulse crowding. Bit shift occurs during recording at moderately high densities. This introduces timing errors in the data transitions during subsequent reads. The clock recovery circuitry interprets these variations as motor speed error, which reduces the system's tolerance to speed variations of the drive motor. This problem has been reduced for the VICTOR 9000 by improving the motor speed control and using an encoding technique that is more tolerant of bit shift error. The disk rotational speed control is accomplished by using a crystal-controlled, closed-loop servo system. The servo system actually consists of two interacting closed servo loops.

The first servo loop is a fast acting inner loop, which is an analog circuit that provides excellent short-term stability. This circuit uses a charge-pump technique, which converts tach pulses from the drive motor to a voltage. This voltage is compared to a reference voltage, and any difference generates a correction in motor speed.

The second servo loop (the outer loop) digitally counts a fixed number of tach pulses from the motor, and measures the period of time that this takes. It then compares this time with the expected time. Any difference results in a modification of the reference voltage for the inner loop. This is accomplished using a single-chip microprocessor (an 8048), which uses the 5 Mhz system clock and two (8 bit) digital-to-analog converters (one per drive). Since this outer loop is crystal-referenced, it provides absolute long term stability and virtually eliminates unit to unit speed differences.

The microprocessor contains a set of speed control tables. These tables are initialized to default values at power-on and are reloadable by the 8088.

7.12 Data Encoding Technique - GCR

To record data on magnetic media, like floppies, the data first has to be converted from the internal computer format into a form that can be stored and retrieved. This is true because data in the internal format may contain long sequences of like bits—either ones or zeroes. If data is recorded with more than a few bit times having no changes (flux reversals), the characteristics of the read channel make it impossible to read back the same signal that was recorded. Also, the data is written at a constant frequency (bit rate), but no clock signal is written. This means that the clock information must be re-created during subsequent read operations. Even though the disk speed is closely controlled (to within 2%), data transitions are required periodically to resynchronize the clock recovery circuitry.

The VICTOR 9000 uses an encoding technique called group code recording (GCR) to convert the data from internal representation to an acceptable form. GCR converts each (4-bit) nibble into a 5-bit code that guarantees a recording pattern that never has more than two zeroes together. Then data is recorded on the disk by causing a flux reversal for each "one" bit and no flux reversal for each "zero" bit.

7.13 Read Channel

The read channel consists of a magnetic pickup (read/write head), an amplifier section, a clock recovery section, a serial to parallel converter, and a 10-bit to 8-bit (GCR to internal form) conversion section.

The read/write head picks up a low amplitude (approximately 2 to 8 millivolts) signal from the disk. This signal is amplified differentially (to minimize the effects of common mode noise), and pass-band filtered (to reduce noise at frequencies other than those of interest). The linear output from the filter is passed to the differentiator, which generates a wave form whose zero crossovers correspond to the peaks of the read signal (these peaks occur approximately where the flux reversals take place during the write). Then this signal is fed to the comparator and digitizer circuitry. The comparator and digitizer circuitry generate a 1-microsecond read data pulse, corresponding to each peak of the read signal. These pulses serve two purposes: first, each of these pulses represents a "one" bit and so sets the serial data latch (to one); second, these pulses are used by the clock recovery circuit to keep a phase-locked loop (PLL) synchronized to the data being read from the disk. At each clock cycle (bit time), the serial data latch is shifted into the serial to parallel converter, and the serial data latch is reset (to zero).

When 10 bits have been shifted into the serial to parallel converter, the data is converted back into the original 8-bit byte. This data byte is latched, and a signal is sent to the processor unit that a byte is ready to be read.

7.14 Write Channel

The write channel consists of an 8 bit to 10 bit (internal form to GCR) code conversion section, a parallel to serial converter, write/erase current control, and the read/write head. The write circuitry is configured so that it is impossible to enable the write current if the diskette is write-protected. The write circuitry also initializes to read mode at power-up, and is prevented from writing until the power has stabilized.

7.15 Sector Format

Figure 7-3 illustrates sector format. Table 7-1 describes the parts of the sector:

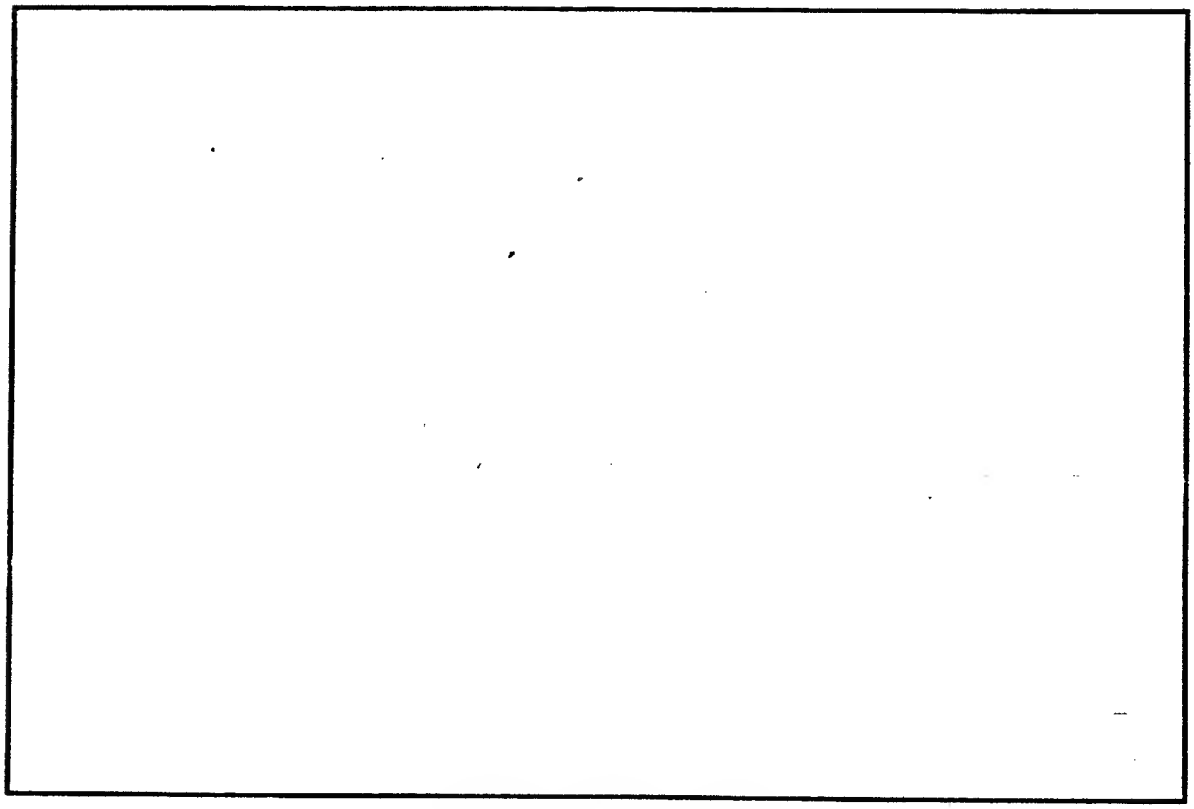

Figure 7-3. Sector Format

COMPONENT	DESCRIPTION
Header sync	This sync mark synchronizes the PLL and causes sync detect interrupts to be sent to the CPU.
Sector header (header ID, track ID, sector ID, and checksum)	This area of 4 bytes contains sector indentification information.
Gap 1	This gap allows time for the CPU to process the sector header information and for the read/write head to clear the header for an update.
Data Sync	This sync mark synchronized the PLL and indicates the start of the data field.
Data field (data sync, data ID, data bytes, and checksum)	This is the useful data content of the sector for error detection if a 2-byte checksum is used.
Gap 2	This gap allows for speed variation during an update so that the next sector sync mark is not overwritten.

Table 7-1. Sector Components

7.16 Track Format

Table 7-2 presents track format:

ZONE NUMBER	TRACK NUMBERS LOWER HEAD (STANDARD)	UPPER HEAD (OPTIONAL0	SECTORS PER TRACK	ROTATIONAL PERIOD (MS)
0	0-3	(unused)	19	237.9
1	4-15	0-7	18	224.5
2	16-26	8-18	17	212.2
3	27-37	19-29	16	199.9
4	38-48	30-40	15	187.6
5	49-59	41-51	14	175.3
6	60-70	52-62	13	163.0
7	71-79	63-74	12	149.6
8	unused	75-79	11	144.0

Table 7-2. Track Format

7.17 Physical Bus Interface

The disk drive interface board connects to the CPU board via a 50 pin ribbon cable. This cable carries the data bus, address lines, and control signals needed to interface to the three 6522's on the interface board. All the I/O ports of the CPU System are memory-mapped, allowing more efficient I/O operations.

8. Expansion Bus Definition

The Expansion Bus is basically a buffered extension of the system's 8088 processor plus additional control and timing signals required to interface the system. The expansion bus consists of

- ▶A multiplexed buffered data bus, BD0 - BD7
- ▶A buffered address bus, A8 - A19
- ▶Various timing, control, interrupt, and power lines

Expansion Bus Pin Definitions

PIN	SIGNAL	I/O	DESCRIPTION
50	A19	I/O	Buffered address bits 8 to 19: These lines are driven from the 8088 during normal operation and are valid from the falling edge of ALE to the rising edge of the next ALE. If an external device takes control of the system via HOLD and HOLD ACKNOWLEDGE, these lines are tri-stated.
1	A18	I/O	
49	A17	I/O	
2	A16	I/O	
48	A15	I/O	
3	A14	I/O	
47	A13	I/O	
4	A12	I/O	
46	A11	I/O	
5	A10	I/O	
45	A09	I/O	
6	A08	I/O	
29	BD7	I/O	Time multiplexed buffered address/data bus: During normal operation, the lower 8 bits of address, AD0–AD7, are valid on the falling edge of ALE.
22	BD6	I/O	
28	BD5	I/O	
23	BD4	I/O	
27	BD3	I/O	
24	BD2	I/O	
26	BD1	I/O	
25	BD0	I/O	
9	ALE	O	Buffered Address Latch Enable: Processor signal which indicates BD0–BD7 contain valid addresses. Typically used to latch low order 8 bits of address.
11	RD	O	Buffered Read Strobe: Processor signal indicating a read cycle.

Table 8-1 Expansion Bus Pin Definitions

PIN	SIGNAL	I/O	DESCRIPTION
14	$\overline{WR}$	O	Buffered Write Strobe: Processor signal indicating a "write cycle."
8	$\overline{DEN}$	O	Buffered Data Enable: Provided by the processor for use as an enable for transceivers.
33	$\overline{DLATCH}$	O	Data Latch: The falling edge of this signal may be used to strobe data generated from a processor read access.
30	$\overline{EXTIO}$	I	External IO: Control line which prevents internal data bus buffers from conflicting with external buffers when mapping external I/O into address space E0000 to EFFFF. "CSEN" should be used as a control signal to disable internal buffers via EXTIO and enable external buffers if using address space E0000 to EFFFF. Addresses used by the system cannot be disabled by EXTIO.
19	CSEN	O	Chip Select Enable: This line is synchronized to phase 2. It is true from a falling edge of phase 2 to the next falling edge of phase 2, when address space E0000 to EFFFF is accessed.
40	CLK15B	O	15-Mhz clock: Signal from which all system timing is derived. Its period is 66.6 nanoseconds with a 50% ±10% duty cycle.
38	CLK5	O	5-Mhz clock: Signal is in phase with the 8088 clock input. Its period is 200 nanoseconds with a 33% duty cycle.
20	PHASE2	O	1-Mhz clock: Signal is asynchronous with CLK5. Its period is 1 microsecond with a 40/60% duty cycle. Useful to interface 6800 type I/O circuits.
21	XACK	I	External Acknowledge: This line is normally high and may be pulled low by external devices resulting in pulling the 8088 Ready input low, generating wait states. This line is resynchronized by the system logic.
17	$\overline{HOLD}$	I	Input to the 8088. This is an external request for control of the system buses.

Table 8-1 Expansion Bus Pin Definitions (*cont.*)

PIN	SIGNAL	I/O	DESCRIPTION
18	HLDA	O	Buffered Hold Acknowledge: System response to "HOLD" request. When true (high) the following signals are tri-stated: A8-A19, IO/$\overline{M}$ BD0–BD7, $\overline{DEN}$ ALE, $\overline{SSO}$ $\overline{RD}$, $\overline{INTA}$ $\overline{WR}$, DT/$\overline{R}$ $\overline{DLATCH}$ is controlled by external logic.
41	READY	O	Status Line: This line reflects the syncronized "ready" input to the 8088.
10	IO/$\overline{M}$	O	Buffered 8088 Status Line: Distinguishes between a memory or I/O bus cycle.
7	$\overline{SSO}$	O	Buffered 8088 status line.
12	DT/$\overline{R}$	O	Buffered Data Transmit/Receive: Processor signal typically used to control the direction of system transceivers. The combination of IO/$\overline{M}$, DT/$\overline{R}$, and SSO provide current bus cycle status: IO/$\overline{M}$ DT/$\overline{R}$ $\overline{SSO}$ DESCRIPTION 0 0 0 Instruction fetch 0 0 1 Read from memory 0 1 0 Write from memory 0 1 1 Passive (no bus cycle) 1 0 0 Interrupt acknowledge 1 0 1 Read from I/O 1 1 0 Write to I/O 1 1 1 Halt
15	$\overline{NMI}$	I	Non-Maskable Interrupt: An edge-triggered input which causes a type-2 interrupt. A transition from high to low initiates the interrupt at the end of the current instruction.
16	IRQ	I	Interrupt Request: This input should be driven with an open collector driver; it is "collector OR'ed" with five 6522's and one 6852 and is pulled to +5 volts through a 3.3Kohm resistor. A low level on any of these circuits generates a high level input to the system 8259 at IR3 level.
43	IR4	I	Interrupt Request Level 4: Direct access to IR4 of the system 8259.

Table 8-1 Expansion Bus Pin Definitions (*cont.*)

PIN	SIGNAL	I/O	DESCRIPTION
42	IR5	I	Interrupt Request Level 5: Direct access to IR5 of the system 8259.
13	RESET	0	System Reset: Generated at power on or from the Reset switch.
44	Ground		
39	Ground		
35	Ground		
31	Ground		
37	+5volts		250ma expansion board
36	+5volts		250ma expansion board
34	+12 volts		50ma expansion board
32	−12 volts		

Table 8-1 Expansion Bus Pin Definitions

SIGNAL	NORMAL USAGE (I/O)	INTERNAL LOAD	EXTERNAL DRIVE
	Tri-stated Lines		
A8-19	O	4	4
BDO-7	I/O	5	4
ALE	O	5	4
RD	O	4	4
WR	O	4	4
DEN	O	4	4
IO/M	O	2	4
SSO	O	1	4
DT/R	O	4	4
	TTL Outputs		
DLATCH	O	-	4*
CSEN	O	-	4*
C1K15B	O	-	1*
C1K5	O	-	4*
Phase 2	O	-	1*
HLDA	O	-	1*
READY	O	-	4
RESET	O	-	4

* If required, buffer through one common IC package, such as 74LS04.

Table 8-2 Expansion Bus Loading

NOTE: All loads are 74LSXX loads of .4ma. External drive, as specified, is for each of the four slots available. Care must be taken to ensure adequate drive for other expansion modules which may be installed in the system.

SIGNAL	INTERNAL LOAD	PULLUP PROVIDED
EXTIO	2	2.2K
XACK	1	2.2K
HOLD	1	2.2K
NMI	1	2.2K
IRQ	1	3.3K

Table 8-3 Inputs Driven with Open Collector Driver

IR4
IR5

Table 8-4 Inputs Direct to System 8259

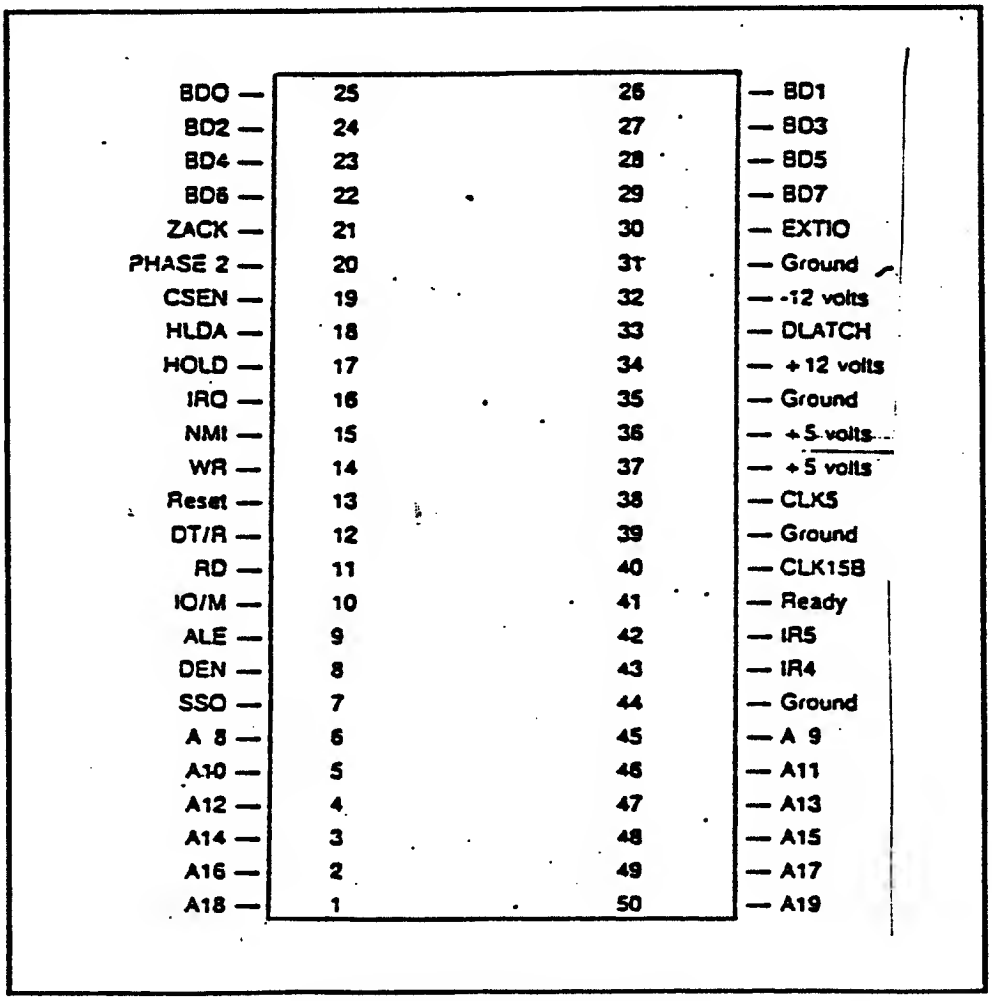

Figure 8-1 Expansion Connector

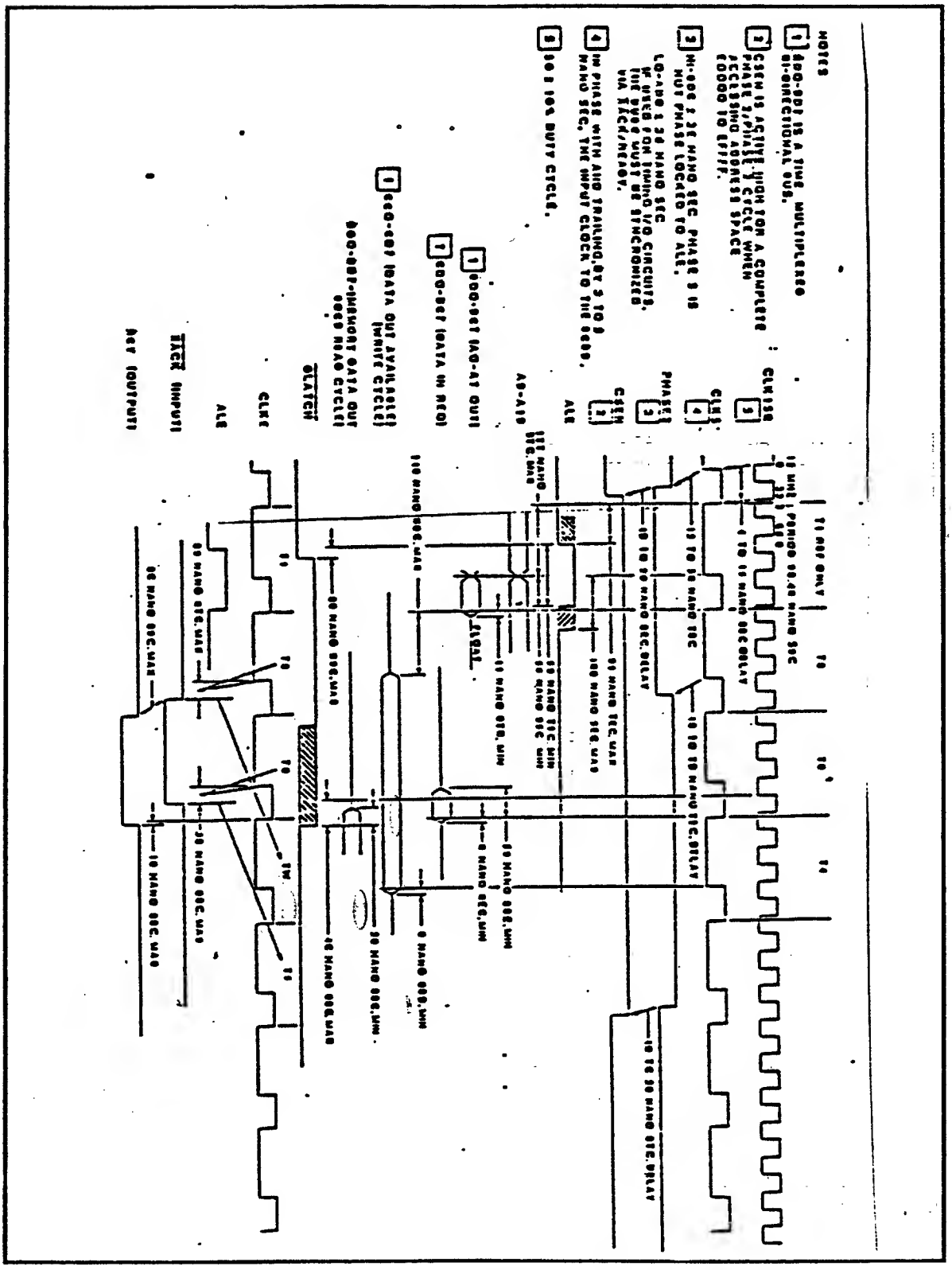

Figure 8-2 Expansion Bus Interface Timing

9. Display System

9.1 Introduction

The VICTOR 9000 display hardware is a memory mapped raster scan system. The display RAM physically occupies 4K bytes, starting at $F0000_{16}$, plus from 4K to 40K bytes of the lower 128K bytes in the 8088 memory map. The display RAM is organized in two separate banks, which operate in a pipelined fashion (see Figure 9-1). The first bank is the screen buffer; it contains the characters which are to be displayed on the screen. The screen buffer also contains attribute information for each character location. The character selection code (called the font cell pointer), together with the character row number (0-15), is used as the address for the second bank, which contains patterns for the characters (font cells). To generate video, the font cell patterns are accessed and latched into the video shift register.

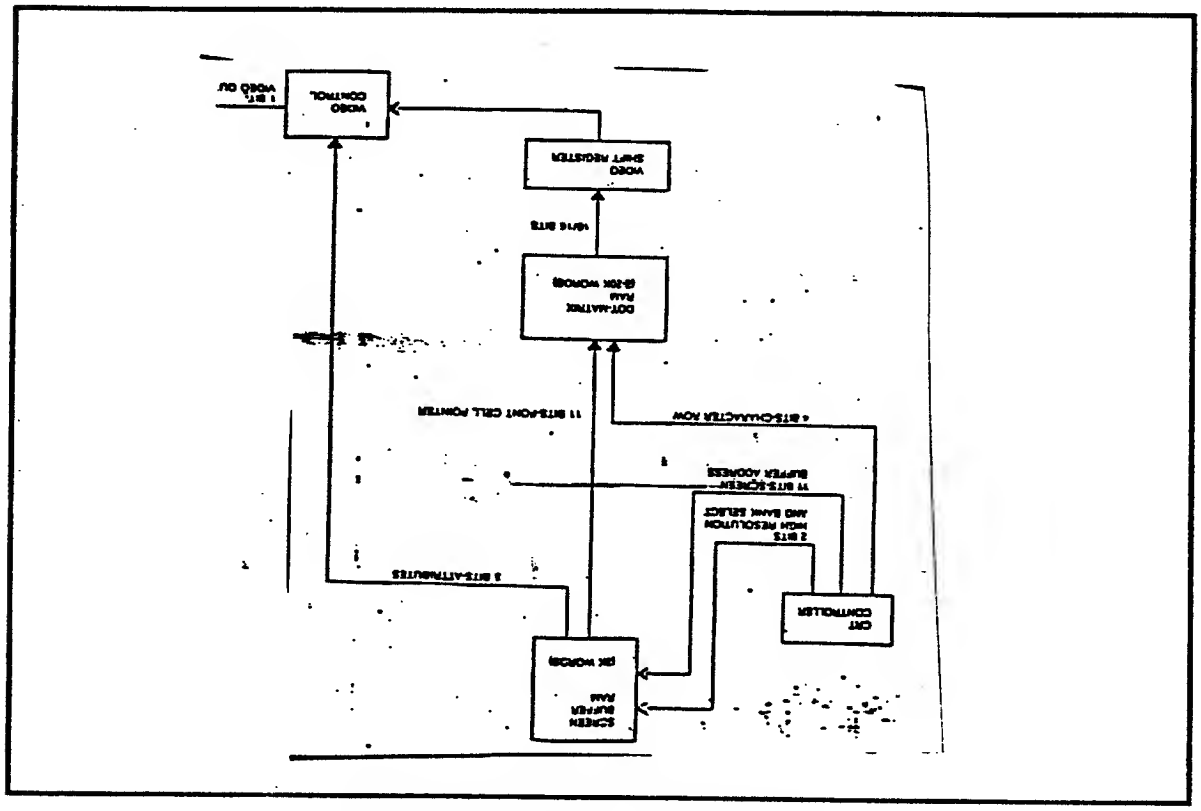

Figure 9-1 Display System Organization

The display hardware is capable of 80 columns by 25 lines of text. The text character cells are 10 dots wide by 16 lines high. These character cells are RAM mapped and programmable. There is also a 5 bit attribute code associated with each character. Four of these attribute bits are used for reverse video, underline/strikeover, highlight, and nondisplay. The other bit is available for user software or external hardware. The display hardware can also be configured for a high resolution mode: 800×400 dots of bit addressable display. In this mode, the reverse video, double intensity,

and nondisplay attributes apply to fixed (16×16 dot) cells on the screen, and the underline/strikeover attribute is not operative.

The character and attribute bits are organized into words called the screen buffer. The lower 11 bits of each word define which of the 2048 possible characters is to be placed at that location of the screen. These 11 bits are collectively called the font cell pointer. The upper five bits of the word are the attributes. The MSB (bit 15) is the reverse video bit. Bit 14 is the low intensity bit. Bit 13 is the underline bit. Bit 12 is the nondisplay bit. The remaining bit (11) is uncommitted.

The screen buffer words are on even address boundaries. The physical memory of the screen buffer is located, in system address space, at $F0000_{16}$ to $F0FFF_{16}$. The 80 character by 25 line display occupies 2000 words (4000 bytes) of the available 2048 words in the screen buffer. Logically, the screen buffer is mapped to include locations $F0000_{16}$ to $F1FFF_{16}$. Therefore, addressing location $F0000_{16}$ accesses the same physical word as addressing location $F1000_{16}$.

The logical beginning of the display screen is selected by a pair of registers in the CRT controller chip (this is a word address). This register pair may be programmed to move the starting address (line one, column one) of the display to any word of the screen buffer. When the control register pair is used in this manner, the screen buffer functions as a 2048 word circular buffer. Using this technique, line scrolling in the text mode may be accomplished by adding 80 to the contents of the screen start register and blanking the 80 words following the previous end of screen. In both these operations, to keep the address within the screen buffer address space, it is also necessary to logically AND the resulting address with $F1FFF_{16}$.

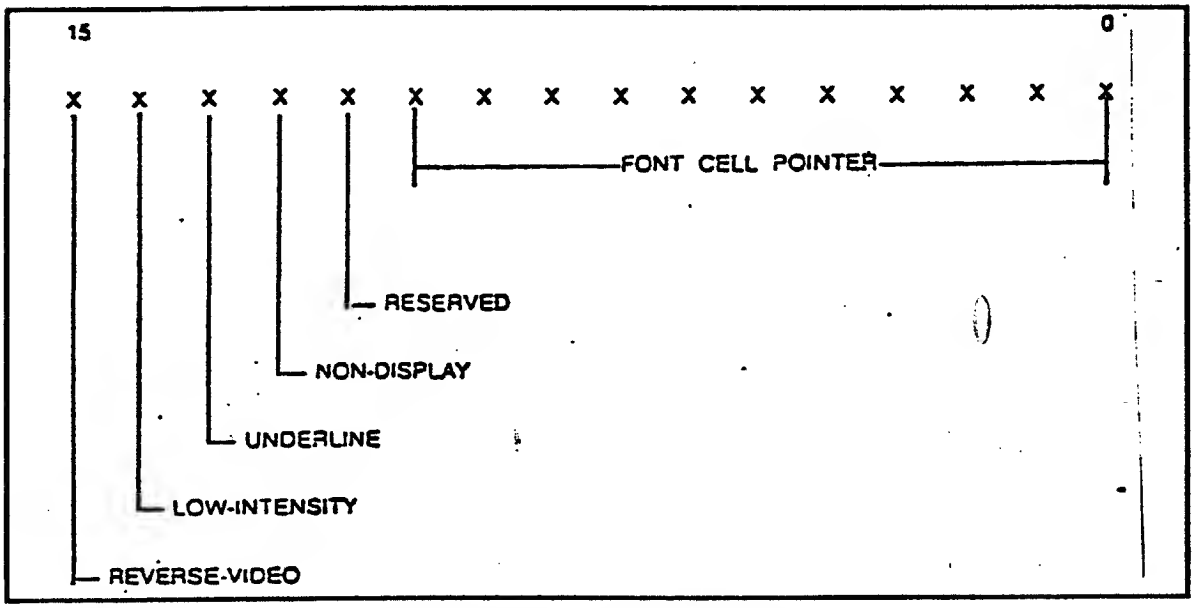

Figure 9-2 Screen Buffer Word Format

The actual dot patterns of each character are stored in the font cell memory. Each 10×16 line character cell is stored in 16 consecutive words. This group of 16 words is called a font cell. The lower 10 bits of each word contain the 10 dots of a scan line of the character picture. The upper left bit of a character would be the LSB of the first word in the 16 consecutive words that define a font cell. Bit 15 of each font cell word is reserved for the underline/strikeover flag bit (in text mode, only). If bit 15 is set and the underline/strikeover attribute (bit 13) from the screen buffer is set, then that scan line will be white; otherwise, the lower 10 bits in that word will be displayed. The nondisplay bit can be used to create "secret" (nondisplayed) characters or fields. If a minimum (128 character)

character set is defined, the font cells would occupy 4K bytes of memory. The font cells can be located anywhere within the first 128K bytes of RAM, but may not cross the 64K boundary.

9.2 High Resolution Mode

The 800×400 dot, bit mapped, high resolution display is a special case use of the cell graphics. The output line, called HIRES (from the CRT controller), controls the character cell width. When this line is high, the character cells are 16 dots wide instead of the usual 10 dots. The screen is then organized as 50 columns by 25 lines of 16×16 dot font cells. This is accomplished by writing new values into the control registers of the CRT controller. The full 16 bits of each font cell word are used to describe the picture of each character. The screen buffer is organized so that each of the 1250 characters on the screen is a different character, as described earlier in this manual. High resolution software then operates directly on the font cell memory for display bit manipulation.

> **Programming Note:** The HIRES/TEXT control and the DOTSEL control (which select whether the beginning address of the font cell memory is to be in the first or the second 64K of system memory) are manipulated via the two high order address bits in the CRTC display address register pair, R12 and R13. This address interacts with the cursor register pair, R14 and R15, and the light pen register pair, R16 and R17. Specifically, if the light pen register pair is used and/or the cursor display function is desired, then the software must: (1) add the cursor address to the current settings of HIRES/TEXT and DOTSEL and (2) subtract or mask these bits when interpreting a light pen interrupt.

9.3 Brightness and Contrast Control

The overall display brightness and the contrast between high and low intensity characters are software adjustable.

Brightness may be adjusted to one of eight different levels by setting the brightness control bits (PB2, PB3, and PB4 of the 6522 at $E8040_{16}$) to the binary value corresponding to the desired level. The binary value range from zero to seven selects increasing brightness levels.

The contrast function controls the difference in intensity between highlighted characters and normal intensity characters. Only the intensity of the normal intensity characters is varied by the contrast function. The contrast function selects one of eight levels by setting the binary value of the desired level in the three contrast control bits (PB5, PB6, and PB7 of the 6522 at $E8040_{16}$). A value range of zero to seven selects increasing differences between the normal and highlighted characters, with zero causing no difference.

9.4 Circuit Description

The lower 128K bytes of RAM is a dual port memory system. One port is used by the display hardware to refresh the raster scan display. The other port is used by the 8088 microprocessor for read and write operations. The dual port memory is managed by an arbitrator circuit that guarantees one refresh access to the display RAM every character cell time. The arbitrator circuit adds a wait state to any 8088 memory cycle if this is necessary to isolate it from the display/refresh cycle. This results in an average of one wait state (200nsec) for every five processor memory access cycles. Processor and memory cycles are normally four clock periods (200nsec). This could cause a decrease of approximately 5% in system bus performance. However, due to the 8088 instruction lookahead queue, this decrease in bus performance rarely translates into decreased system performance.

The display refresh addresses are generated by the HD46505S CRT controller chip (CRTC). Of the 14 address lines from the CRTC, 11 (MA0-MA10) are used to address the 2K words of screen buffer RAM. The 16 data lines output by the screen buffer are latched and divided into 11 lines of character address information and 5 lines of character attributes. The attribute bits are sent, via a set of character sync registers, to the video control section. The 11 lines of the character address are combined with 4 lines of character row address and MA12 (DOTSEL) from the CRTC. This address is then multiplexed down to 8 font cell address lines. The 14th character address line (MA13) is used to select the high resolution mode. The 16 bit data output word from each font cell word is latched and sent to a 16 bit shift register. Either 10 or 16 dots of the shift register are shifted out to the video control section. The video control section adds the reverse video, highlight, underline, and nondisplay attribute bits and the cursor output from the CRTC. The result is sent to the video display, along with horizontal and vertical sync pulses.

The display circuit manages the memory refresh in the 128K bytes of on-board dynamic RAM. The horizontal and vertical retrace intervals are used for memory refresh. Display/refresh cycles occuring during retrace intervals cause 8 bits from the refresh address counter to be sent to all 128K of dynamic RAM, rather than the normal display address lines. The display CAS signal is inhibited for a RAS-only memory refresh. The memory- refresh counter is clocked after each refresh cycle. In every 64 microsecond horizontal display period, 15 memory refresh cycles occur. Every 2 ms, 480 memory refresh addresses are generated, exceeding the 128-address-per-2ms specified requirement of 16K dynamic RAM.

9.5 CRTC Device Operation Overview

The CRTC consists of an internal register group, horizontal and vertical timing circuits, a linear address generator, a cursor control circuit, and a light pen detection circuit. Horizontal and vertical timing circuits generate RA0-RA4 , DISPTMG, SYNC, and VSYNC. RA0-RA4 are raster (row) address signals and are used as address bits 1 to 4 for the font cell accesses. DISPTMG, HSYNC, and VSYNC signals are sent to the video control circuit. This horizontal and vertical timing circuit consists of an internal counter and comparator circuit.

The linear address generator generates refresh memory address MA0-MA11 to be used for refreshing the screen. The light pen detection circuit detects the light pen position on the screen. When the light pen strobe signal is received, the light pen register latches the address generated by the linear address generator to save the position of the pen on the screen. The cursor control circuit controls the position of the cursor, its height, and its blink rate.

The CRTC provides 13 interface signals to the CPU and 25 interface signals to the display circuits.

REGISTER	CHARACTER MODE	HIGH RESOLUTION MODE
R0	5C	3A
R1	50	32
R2	51	34
R3	CF	C9
R4	19	19
R5	06	06
R6	19	19
R7	19	19
R8	03	03
R9	0E	0E
R10	60	20
R11	0F	0F
R12	00	00
R13	00	00
R14	00	00
R15	00	00

NOTE: All values are in hexadecimal.

Table 9-1 Recommended Values for CRTC Register Initialization

9.6 Interface Signals to the CPU

Bidirectional Data Bus (ID0-ID7)

The bidirectional data bus is used for data transfer betweeen the CRTC and the 8088. The data bus outputs are 3 state buffers and remain in the high impedance state except when the 8088 performs a CRTC read operation.

Read/Write (R/W)

The R/W signal controls the direction of data transfer between the CRTC and the 8088. When R/W is high, CRTC data is transferred to the 8088. When R/W is low, 8088 data is transferred to the CRTC.

Chip Select (CS)

The CS signal is used to address the CRTC. When CS is low, it enables R/W operation to CRTC internal registers. This signal is derived from decoded address signals of the the 8088.

Register Select (RS)

The RS signal is used to select the address register and the 18 control registers of the CRTC. When RS is low, the address register is selected; when RS is high, control registers are selected. This signal is the lowest bit (A0) of the 8088 address bus.

Enable (E)

The E signal is used as strobe signal in 8088 R/W operations with the CRTC internal registers. This signal is PHASE2.

Reset (RES)

The Reset signal (RES) is an input signal used to reset the CRTC. When RES is low, it forces the CRTC into the following status:

- ▶All the counters in the CRTC are cleared, and the device stops the display operation.
- ▶All the outputs go low
- ▶Control registers in the CRTC are not affected.

9.7 Interface Signals to Display Circuits

Character Clock (CLK)

CLK is a standard clock input signal which defines character timing for the CRTC display operation. This signal is provided by the memory controller.

Horizontal Sync (HSYNC)

HSYNC is an active high level signal which provides horizontal synchronization for the display device.

Vertical Sync (VSYNC)

VSYNC is an active high level signal which provides vertical synchronization for the display device.

Display Timing (DISPTMG)

DISPTMG is an active high level signal which defines the display period in horizontal and vertical raster scanning. It is necessary to enable the video signal only when DISPTMG is high.

Refresh Memory Address MA0-MA13

MA0-MA11 are refresh memory address signals which are used to access the screen buffer in order to refresh the CRT screen periodically.

MA11 is unused.

MA12 selects the 64K memory bank to be used for font cell memory.

When MA12 equals 0, it selects system RAM starting at location 0; when MA12 equals 1, it selects system RAM starting at location 10000_{16}.

When MA13 equals 0, it selects text mode when MA13 equals one, it selects bit mapped HIRES mode.

Raster Address (RA0-RA4)

RA0-RA4 are row address signals which are used to select the row of the current character in the font cell memory to be displayed.

Cursor Display (CUDISP)

CUDISP is an active high level video signal which is used to display the cursor on the CRT screen at the current display location. This output is inhibited while DISPTMG is low. This output is mixed with the video signal and is provided to the CRT display circuits.

Light Pen Strobe (LPSTB)

LPSTB is an active high level input signal which accepts a strobe pulse detected by the light pen and control circuit. When this signal is activated, the memory address (MA0-MA11), along with the current settings of HIRES and DOTADR, are stored in the 14 bit light pen register. The stored memory address needs to be corrected in software, taking the delay time of the display device, light pen, and light pen control circuits into account.

9.8 Internal Registers

Address register (AR)

AR is a 5 bit register used to select among the 18 internal control registers (R0-R17). The value of AR is the address of one of 18 internal control registers. Data values from 18 to 31 do nothing. Access to R0-R17 requires writing the address of the corresponding control register into this register.

Horizontal total register (R0)

The contents of R0 program the total number of horizontal character clock periods per line, including the retrace period. The data is 8 bit, and its value should be programmed according to the selected mode of the display. The programmed value must be one less than the number of character intervals required. When programming for interlace mode, the value must be even.

Horizontal displayed register (R1)

R1 is used to program the number of displayed characters per horizontal line. Data is 8 bit, and any value smaller than that in R0 is valid.

Horizontal sync position register (R2)

The contents of R2 program the horizontal sync position in units of the character clock period. Data is 8 bit, and any value less than R0 is valid. The value programmed should be one less than the sync position desired. The effect of increasing the value in R2 is to shift all characters displayed on the CRT screen to the left. When the value is decreased, character positions shift to the right.

Sync width register (R3)

The contents of R3 set the horizontal sync pulse width and the vertical sync pulse width. The horizontal sync pulse width is programmed in the lower 4 bits, in units of the character clock period (0 is invalid). The vertical sync pulse width is programmed in the upper 4 bits, in units of the horizontal period. When 0 is programmed in the upper 4 bits, 16 horizontal periods are specified.

Vertical total register (R4)

R4 is used to program the total number of horizontal scans per frame, including the vertical retrace period. This is a 7 bit value and should be programmed according to the selected display mode. The programmed value should be one less than the number desired.

Vertical total adjust register (R5)

The contents of R5 select the total number of horizontal scans per field. This register allows fine control of the deflection frequency.

Vertical displayed regster (R6)

R6 is used to determine the number of displayed character rows on the CRT screen. This is a 7 bit value, and any number that is smaller than that in R5 is valid.

VSW				
2^7	2^6	2^5	2^4	PULSE WIDTH*
0	0	0	0	16
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

* Note: Pulse Width is in horizontal periods

Table 9-2 Pulse Width of Vertical Sync Signal

HSW				
2^3	2^2	2^1	2^0	PULSE WIDTH*
0	0	0	0	not used
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

*** Note: Pulse Width is in character clock periods; HSW = 0 cannot be used.**

Table 9-3 Pulse Width of Horizontal Sync Signal

Vertical sync position register (R7)

The contents of R7 set the vertical sync position on the screen, in units of the horizontal character line period. Data is 7 bit, and any number that is equal to or less than the vertical total register can be programmed. The value programmed should be one less than the position desired. Increasing the value shifts the display upward. Decreasing the values shifts the display downward.

Interlace and skew register (R8)

R8 programs the raster scan mode and the skew (delay) of CUDISP and DISPTMG.

Interlace mode program bits (V, S)

The raster scan mode is selected by the V and S bits.

V bit	S bit	RASTER SCAN MODE
0	0	Noninterlace mode
1	0	Noninterlace mode
0	1	Interlace sync mode
1	1	Interlace sync and video mode

Table 9-4 Interlace Mode (D0, D1)

Skeq program bit (C1, C0, D1, D0)

The C1, C0, D1, and D0 bits are used to program the skew (delay) of CUDISP and DISPTMG.

The skews of the two signals are programmed separately.

D1 bit	D0 bit	DISPTMG SIGNAL
0	0	Zero skew
0	1	One character skew
1	0	Two character skew
1	1	No output

Table 9-5 DISPTMG skey bit (D7, D6)

C1 bit	C0 bit	NON SKEW
0	0	Zero skew
0	1	One character skew
1	0	Two character skew
1	1	No output

Table 9-6 Cursor skew bit (D5, D4)

The skew function is used to delay the CUDISP and DISPTMG signals for optimum screen memory access, dot matrix memory, and video signal timing.

Maximum raster address register (R9)

R9 is used to program the maximum raster address (5 bits). This register defines the number of rasters (lines) per character, including intercharacter spaces. Programming is as follows:

▶ Noninterlace Mode

In the following tabulation, the value parameter is set at 4.

RASTER ADDRESS	RESULTING FORMAT
0	————————
1	————————
2	————————
3	————————
4	————————

NOTE: the number of rasters produced in the character format is 5 (one more than the value programmed).

▶ Interlace Sync Mode

In the following tabulation, the value parameter is 4.

RASTER ADDRESS	RESULTING FORMAT
0	————————
0	• • • • • • • • •
1	————————
1	• • • • • • • • •
2	————————
2	• • • • • • • • •
3	————————
3	• • • • • • • • •
4	————————
4	• • • • • • • • •

NOTE: ———————— and • • • • • • • • • denote alternate fields.

The total number of rasters in the character is 10. The number is found by doubling the sum of one plus the value programmed.

► Interlace Sync and Video Mode

The total number of rasters in the character format is one more than the value parameter, as in the noninterlace mode, but the rasters alternate fields. In the following tabulation, a value parameter of 4 is set.

RASTER ADDRESS	RESULTING FORMAT
0	————————
10 • • • • • • • •	
2	————————
30 • • • • • • • •	
4	————————

NOTE: ———————— and • • • • • • • • • denote alternate fields.

Cursor start raster register (R10)

RIO programs the cursor start raster (line) address and the cursor display mode. The lower 5 bits (D0-D4) are cursor start, and the next 2 bits (D5, D6) are cursor mode.

D5	D6	CURSOR DISPLAY MODE
0	0	Steady cursor
0	1	Cursor off
1	0	Blinking cursor, 16 field period
1	1	Blinking cursor, 32 field period

Table 9-7 Cursor Display Mode (D6, D5)

Curser end raster register (R11)

R11 sets the cursor end raster (line) address.

Start address registers (R12, R13)

R12 and R13 are used to program the first (word) address of the screen buffer memory to be displayed. This word will display as line one/column one on the display screen.

Cursor registers (R14, R15)

The two read/write registers R14 and R15 store the cursor location. The upper 2 bits (D6, D7) of R14 must always be set to 0.

Light pen registers (R16, R17)

The read only registers R16 and R17 are used to latch the detection time address of the light pen. The upper 2 bits (D6, D7) of R16 are always 0. The value latched may need to be corrected by software to allow for light pen system delays.

9.9 Restrictions on Programming Internal Registers

The following restrictions on programming internal registers apply:

$0<Nhd<(Nht+1)\leq 256$
$0<Nvd<(Nvt+1)\leq 128$

$0\leq Nhsp\leq Nht$

$0\leq Nvsp\leq Nvt$*

$0\leq NCSTART\leq NCEND\leq Nr$ (noninterlace, interlace sync mode)
$0\leq NCSTART<NCEND\leq Nr+1$ (interlace sync and video mode)

$2\leq Nr\leq 30$

$3\leq Nht$ (except non interlace mode)
$5\leq Nht$ (noninterlace mode only)

* In interlace mode, pulse width is changed +1/2 by the raster time when the vertical sync signal extends over two fields.

NOTES: The values programmed in the internal registers of the CRTC are used directly to control the CRT. Consequently, the display may flicker if the contents of the registers are changed asyncronously to the display operation. The registers should be changed only during the horizontal or vertical retrace period.

9.10 Noninterlace Mode Display

Alternate fields are identical. The values of raster addresses (RA0-RA4) are counted, starting at zero.

9.11 Interlace Sync Mode Display

In the interlace sync mode, raster addresses in the even field and the odd field are the same. The same character pattern is displayed in both fields with the displayed position in the odd field 1/2 raster space down from that in the even field.

9.12 Interlace Sync and Video Mode Display

In interlace sync and video mode, when the raster number is even, the output raster address is different from when the raster number is odd.

REGISTER	REGISTER NAME	VALUE
R0	Horizontal total	Nht
R1	Horizontal displayed	Nhd
R2	Horizontal sync position	Nhsp
R3	Sync width	Nvsw, Nhsw
R4	Vertical total	Nvt
R5	Vertical total adjust	Nadj
R6	Vertical displayed	Nvd
R7	Vertical sync position	Nvsp
R8	Interlace and skew	
R9	Maximum raster address	Nn
R10	Cursor start raster	
R11	Cursor end raster	
R12	Start address (H)	0
R13	Start address (L)	0
R14	Cursor (H)	
R15	Cursor (L)	
R16	Light pen (H)	
R17	Light pen (L)	

NOTE: Nhd<Nht, Nvd<Nvt

Table 9-8 Programmer Values into the Registers

TOTAL NUMBER OF RASTERS IN THE CHARACTER FORMAT		EVEN FIELD	ODD FIELD
Even		Even Address	Odd Address
Odd			
	Even Line	Even Address	Odd Address
	Odd Line	Odd Address	Even Address

NOTE: Internal line address begins from zero.

Table 9-9 Output Raster Address in Interlace Sync and Video Mode

NOTE: A wide disparity in the number of ON dots in even fields versus that in odd fields causes unequal average beam currents during alternate fields. This causes CRT final anode voltage to differ during alternate fields. Since the deflection factor is a function of this voltage, the two fields will have somewhat different widths. Characters will be distorted, particularly near the edges of the screen. Programming for an odd number of rasters per character line is a good way to reduce this type of problem.

9.13 Cursor Control

Figure D-3 shows display patterns in which various values are stored in the cursor-start-raster register and the cursor-end-raster register. Values in the cursor-start-raster register and the cursor-end-raster register must meet the following conditions:

cursor-start-raster≤cursor-end-raster≤maximum-raster-address

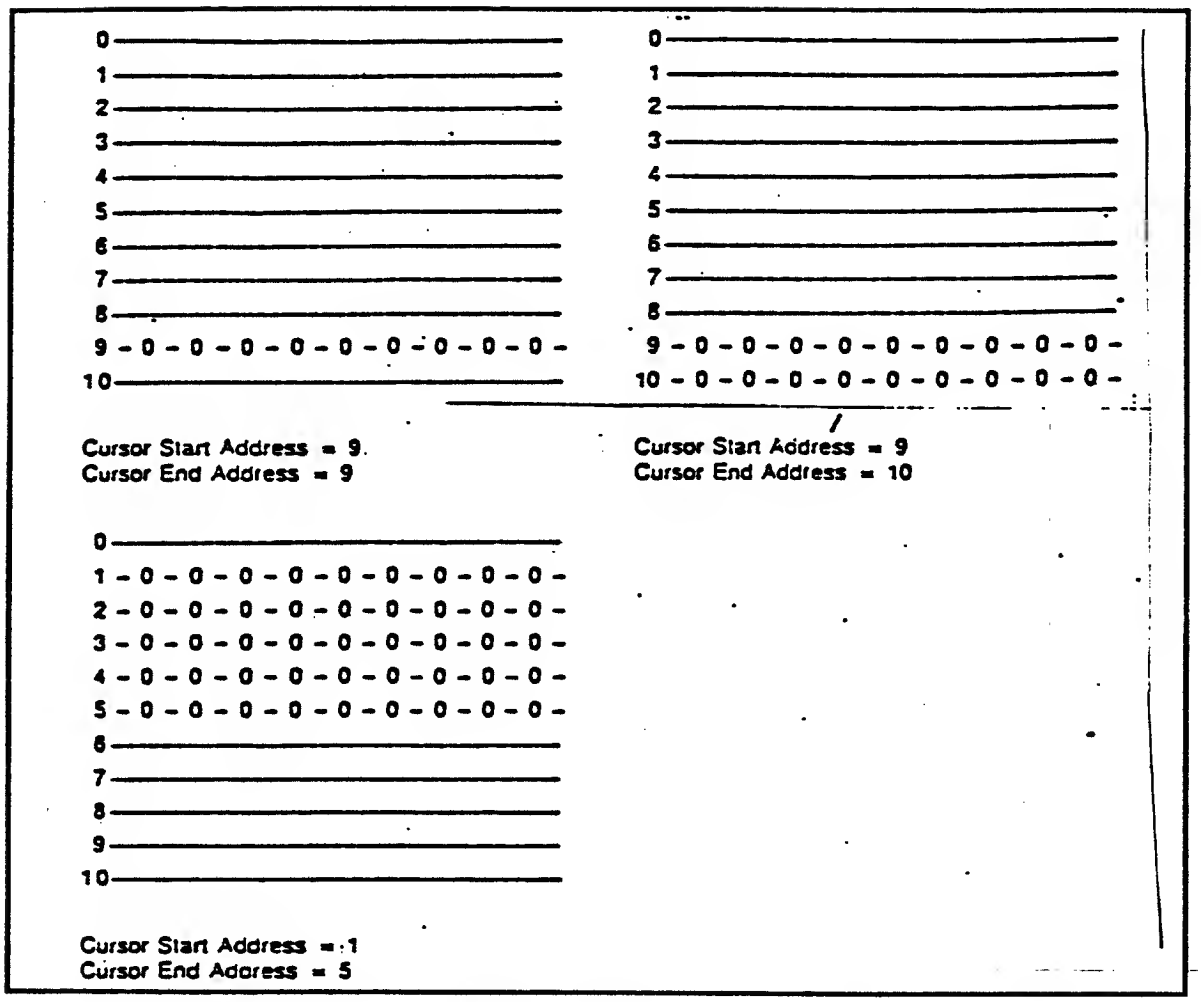

Figure 9-3 Cursor Control

10. I/O Addresses and Bit Assignments

INTERRUPT LEVEL	SIGNAL NAME	DESCRIPTION
IR0	SYN	Sync detect
IR1	COMM	Serial communications
(7201)		
IR2	TIMER	8253 Timer
IR3	PARALLEL	All 6522 IRQ (including disk)
IR4	IR4	Expansion IR4
IR5	IR5	Expansion IR5
IR6	KBINT	Keyboard data ready
IR7	VINT	Vertical sync or nonspecific interrupt

Table 10-1 8259A (PIC IOD0) Address: E0000 - E0001

I/O NAME	SIGNAL NAME	DESCRIPTION
CLK2	100KHZ	Clock input (for time of day)
GATE2	+5V	
OUT2	TIMER	Interrupt for time of day
GLK1	2.5MHZ	Clock input for serial port
GATE1	+5V	
OUT1	MUX SERIAL B	To serial port B MUX
CLK0	2.5MHZ	Clock input for serial port A
GATE0	+5V	
OUT0	MUX SERIAL A	To serial port A MUX

Table 10-2 8253 (TIMER - IOD1) Address: E0020 - E0023

I/O NAME	SIGNAL NAME	DESCRIPTION
RXCA	J8-17	Receive clock A
TXCA	J8-15	Transmit clock A
RXDA	J8-3	Receive data A
TXDA	J8-2	Transmit data A
CTSA	J8-5	Clear to send A
RTSA	J8-4	Request to sendA
DCDA	J8-8	Data carrier detect A input
DTRA	J8-20	Data terminal ready A
RXCB	J9-17	Receive clock B
TXCB	J9-15	Transmit clock B
RXDB	J9-3	Receive data B
TXDB	J9-2	Transmit data B
CTSB	J9-5	Clear to send B
RTSB	J9-4	Request to sendB
DCDB	J9-8	Data carrier detect B input
DTRB	J9-20	Data terminal ready B

Table 10-3 7201(COMM.CTLR IOD2) Address: E0040 - E0043

INTERRUPT LEVEL	SIGNAL NAME	DESCRIPTION
MA13	HIRES	Hires enable output
MA12	DOT ADDR	32-K-WORD page select output (1 = UPPER)

Table 10-4 HD46505S (CRTC CSO) Address: E8000 - E8001

I/O NAME	SIGNAL NAME	DESCRIPTION
PA0	DIO1	Parallel data bit 0, IN/OUT
PA1	DIO2	Parallel data bit 1, IN/OUT
PA2	DIO3	Parallel data bit 2, IN/OUT
PA3	DIO4	Parallel data bit 3, IN/OUT
PA4	DIO5	Parallel data bit 4, IN/OUT
PA5	DIO6	Parallel data bit 5, IN/OUT
PA6	DIO7	Parallel data bit 6, IN/OUT
PA7	DIO8	Parallel data bit 7, IN/OUT
CA1	NRFD	Parallel NRFD interrupt IN
CA2	NDAC	Parallel NDAC interrupt IN
PB0	DAV	Parallel DAV IN/OUT
PB1	EOI	Parallel EOI, IN/OUT
PB2	REN	Parallel REN, IN/OUT
PB3	ATN	Parallel ATN, IN/OUT
PB4	IFC	Parallel IFC, IN/OUT
PB5	SRQ	Parallel SRQ, IN/OUT
PB6	NRFD	Parallel NRFD, IN/OUT
PB7	NDAC	Parallel NDAC, IN/OUT
CB1	N.C.	
CB2	CODEC VOL	Pulse width control codec volume output (TZ)

Table 10-5 6522 (VIA 1 CS1) Address: E8020 - E802F

I/O NAME	SIGNAL NAME	DESCRIPTION
PA0	INT/EXTA	Serial A clock select (LOW = INT)
PA1	INT/EXTB	Serial B clock select (LOW = INT)
PA2	RIA	Serial A ring indicate (J8-22)
PA3	DSRA	Serial A data set ready (J8-6)
PA4	RIB	Serial B ring indicate (J9-22)
PA5	DSRB	Serial B data set ready (J9-6)
PA6	KBDATA	Data from keyboard
PA7	VERT	Vertical signal input (from CRTC)
CA1	N.C.	
CA2	SRQ/BUSY	Parallel port IN/OUT
PB0	TALK/LISTEN	Parallel port direction, control, OUTPUT
PB1	KBACKCTL	Keyboard acknowledge, control, OUTPUT
PB2	BRT0	LSB of brightness control, OUTPUT
PB3	BRT1	Intermediate bit of brightness control, OUTPUT
PB4	BRT2	MSB of brightness control, OUTPUT
PB5	CONT0	LSB of contrast control, OUTPUT
PB6	CONT1	Intermediate bit of contrast control, OUTPUT
PB7	CONT2	MSB of contrast control, OUTPUT
CB1	KBRDY	Key data ready, INPUT
CB2	KBDATA	Shift register INPUT

Table 10-6 6522 (VIA 2 CS2) Address: E8040 - E804F

I/O NAME	SIGNAL NAME	DESCRIPTION
RXCLK		Inverted input from PB7 of VIA3 (codec clock)
TXCLK		Inverted input from PB7 of VIA3 (codec clock)
RXDATA		Input digital data from codec
TXDATA		Digital data output to codec
SM/DTR		Encode/decode control for codec (LOW = DECODE, or TRANSMIT)
DCD		Inverted input from SM/DTR of this chip
CTS		Input from SM/DTR of this chip

Table 10-7 6852 (SSDA CS3) Address: E8060 - E806F

I/O NAME	SIGNAL NAME	DESCRIPTION
PA0	J5-16	Control port
PA1	J5-18	Control port
PA2	J5-20	Control port
PA3	J5-22	Control port
PA4	J5-24	Control port
PA5	J5-26	Control port
PA6	J5-28	Control port
PA7	J5-30	Control port
CA1	J5-12	Control port
CA2	J5-14	Control port
PB0	J5-32	Control port
PB1	J5-34	Control port
PB2	J5-36	Control port
PB3	J5-38	Control port
PB4	J5-40	Control port
PB5	J5-42	Control port
PB6	J5-44	Control port
PB7	J5-46	Codec clock output
CB1	J5-48	Control port
CB2	J5-50	Control port

Table 10-8 6522 (VIA 3 CS4) Address: E8080 - E808

I/O NAME	SIGNAL NAME	DESCRIPTION
PA0	L0MS0	Drive 0 motor speed, OUTPUTS*
PA1	L0MS1	
PA2	L0MS2	
PA3	L0MS3	
PA4	ST0A	Drive 0 spepper phase, OUTPUTS
PA5	ST0B	
PA6	ST0C	
PA7	ST0D	
CA1	DS0	Door 0 sense interrupt, INPUT
CA2	MODE	Write sync
PB0	L1MS0	Drive 1 motor speed, OUTPUTS
PB1	L1MS1	
PB2	L1MS2	
PB3	L1MS3	
PB4	ST1A	Drive 1 spepper phase, OUTPUTS
PB5	ST1B	
PB6	ST1C	
PB7	ST1D	
CB1	DS1	Door 1 sense interrupt, INPUT
CB2	N.C.	

* also used as a data bus to load 8048 parameters during motorspeed controller initialization.)

Table 10-9 6522 (VIA 4 CS5) Address: E80A0 - E80AF

I/O NAME	SIGNAL NAME	DESCRIPTION
PA0	LED0A	LED, drive A, OUTPUT
PA1	TRK0D0	TRACK 0, drive A SENSE, INPUT
PA2	LED1A	LED, drive B, OUTPUT
PA3	TRK0D1	Track 0, drive B SENSE, INPUT
PA4	SIDE SELECT	Dual side select, OUTPUT
PA5	DRIVE SELECT	Select drive A/B, OUTPUT
PA6	WPS	Write protect sense, INPUT
PA7	SYNC	Disk sync detect, INPUT
CA1	GCRERR	GCR error INPUT
CA2	DRW	Disk read/write control, OUTPUT
PB0*	RDY0	Motor speed status, drive A
PB1*	RDY1	Motor speed status, drive B
PB2	SCRESET	Motor speed controller (8048) reset, OUTPUT
PB3	DS1	Door B sense, INPUT
PB4	DS0	Door A sense, INPUT
PB5	SINGLE/$\overline{\text{DOUBLE}}$ sided, INPUT	
PB6	Stepper enable A	
PB7	Stepper enable B	
CB1	N.C.	
CB2	ERASE	Erase head ON/OFF, OUTPUT

* Also used as handshake lines during speed controller initialization.

Table 10-10 6522 (VIA 6 CS6) Address: E80C0 - E80CF

I/O NAME	SIGNAL NAME	DESCRIPTION
PA0	E0	
PA1	E1	
PA2	I2	
PA3	E2	Disk data INPUT
PA4	E4	
PA5	E5	
PA6	I7	
PA7	E6	
CA1	BRDY	Byte ready INPUT
CA2	RDY0	Motor speed status interrupt, drive 0
PB0	WD0	
PB1	WD1	
PB2	WD2	
PB3	WD3	Disk data OUTPUT
PB4	WD4	
PB5	WD5	
PB6	WD6	
PB7	WD7	
CB1	N.C.	
CB2	RDY1	Motor speed status interrupt, DRIVE 1

Table 10-11 6522 (VIA 5 CS7) Address: E80E0 - E80EF

11. Addressing Modes

The 8088 accesses instruction operands in many different ways. Operands can be in registers, instructions, memory, or I/O ports. Memory address and I/O port operands can be calculated several ways. These addressing modes extend the flexibility and convenience of the instruction set. This section briefly describes register and immediate operands, and then covers the 8088 memory and I/O addressing modes in detail.

11.1 Register and Immediate Operands

The quickest, most compact executing instructions specify only register operands. This is because register address is encoded in instructions in a very few bits, and the operation is performed entirely within the CPU (no bus cycles are run). Registers can be source operands and/or destination operands.

Immediate operands are constant data 8- or 16-bits long, contained in an instruction that is available directly from the instruction queue and can be accessed quickly. Like a register operand, no bus cycles are needed to obtain an immediate operand. Immediate operands are limited; they are constant values and can only serve as source operands.

11.2 Memory Addressing Modes

Memory operands must be transferred to or from the CPU over the bus. The EU has direct access to register and immediate operands. When the EU needs to read or write a memory operand, it passes an offset value to the BIU. The BIU adds the offset to the (shifted) content of a segment register, producing a 20-bit physical address. Then it executes the bus cycle(s) needed to access the operand.

11.3 Effective Address

The operand's effective address (EA) is the offset calculated by EU for a memory operand. EA is an unsigned 16-bit number expressing the operand's distance in bytes from the beginning of the segment in which it resides.

The EU calculates the EA in several different ways. Information encoded in the second byte of the instruction tells the EU how to calculate the EA of each memory operand. A compiler or assembler derives this information from the statement or instruction written by the programmer. Assembly language programmers have access to all addressing modes.

Figure 11-1 shows that the execution unit calculates the EA by adding a displacement, the content of a base register, and the content of an index register. The variety of 8088 memory addressing modes results from combinations of these three components in a given instruction.

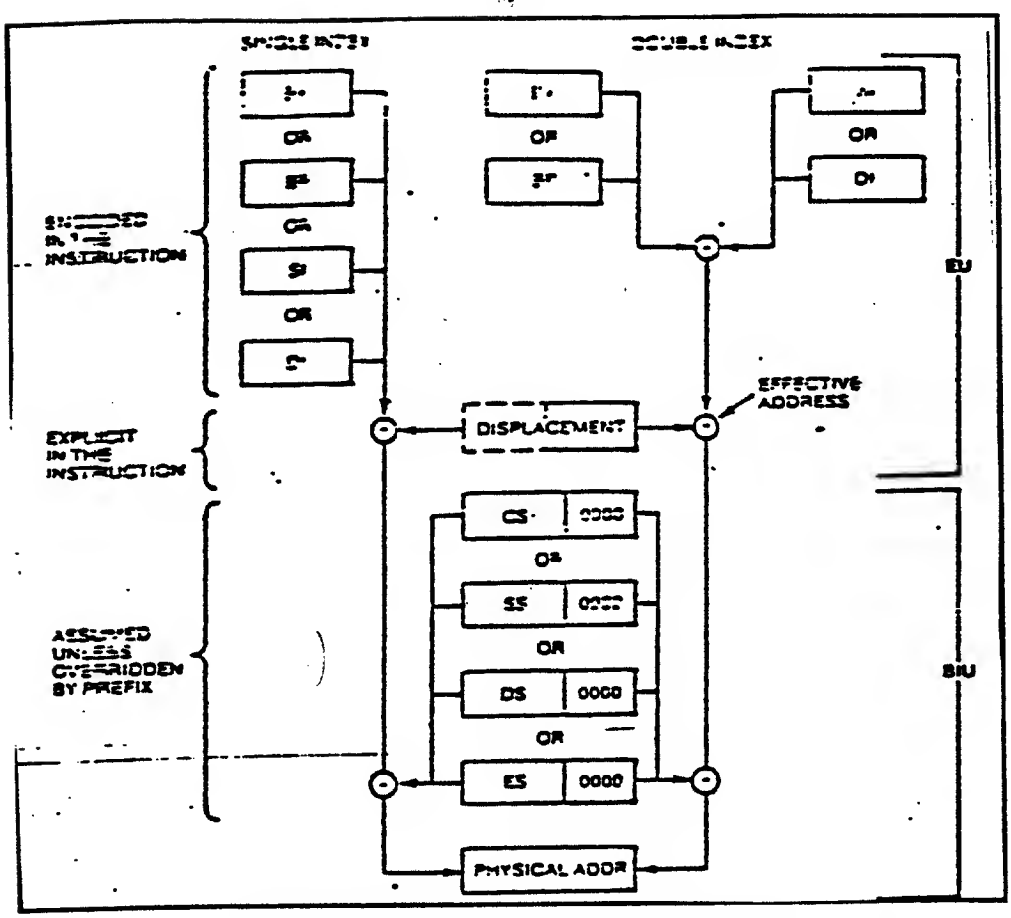

Figure 11-1. Memory Address Computation

The displacement, an 8 or 16-bit number contained in the instruction, is derived from the position of the operand name (a variable or label) in the program. A programmer can modify this value or specify the displacement.

A programmer can specify that BX or BP serve as a base register whose content is to be used in the EA computation. SI or DI can be specified as an index register. The displacement value can change the contents of the base and index registers can change during execution. This makes it possible for one instruction, as determined by current values in the base and/or index registers, to access different memory locations.

It takes time for EU to calculate a memory operand's EA. The more elements in the calculation, the longer it takes. Table 11-1 shows the time required to compute an effective address for any combination of displacement, base register, and index register.

EA COMPONENTS		CLOCKS*
Displacement Only		6
Base or Index Only	BX,BP,SI,DI	5
Displacement + Base or Index	BX,BP,SI,DI	9
Base + Index	BP+DI,BX+SI	7
	BP+SI,BX+DI	8
Displacement + Base + Index	BP+DI+DISP	11
	BX+SI+DISP	11
	BP+SI+DISP	12
	BX+DI+DISP	12

*Add 2 clocks for segment override.

Table 11-1. Effective Address Calculation Time

11.4 Direct Addressing

Direct addressing (see Figure 11-2) is the simplest memory addressing mode. No registers are involved; the EA is taken directly from the displacement field of the instruction. Direct addressing is used to access simple variables (scalars).

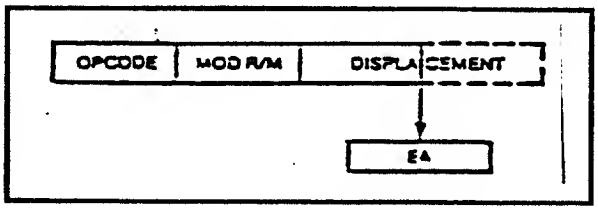

Figure 11-2. Direct Addressing

11.5 Register Indirect Addressing

The effective address of a memory operand can be taken from one of the base or index registers, as shown in Figure 11-3. When the value in the base of the index register is updated appropriately, one instruction can operate on many different memory locations. The load effective address (LEA) and arithmetic instructions change the register value.

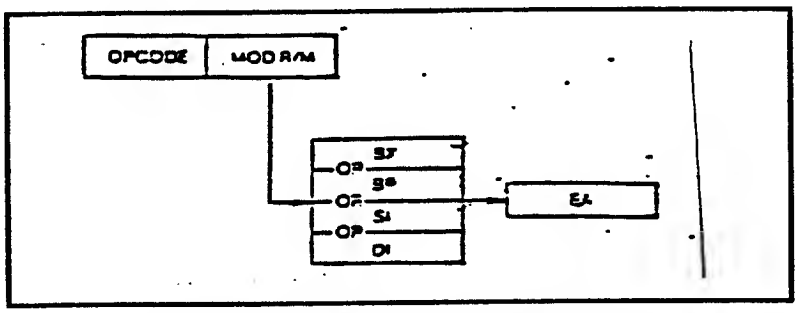

Figure 11-3. Register Indirect Addressing

NOTE: Any 16-bit general register can be used for register indirect addressing with the JMP or CALL instructions.

11.6 Based Addressing

In based addressing (Figure 11-4), the effective address is the sum of a displacement value and the content of register BX or register BP. Specifying BP as a base register directs the BIU to obtain the operand from the current stack segment (unless a segment override prefix is present). Therefore, based addressing with BP is a convenient way to access stack data.

Based addressing provides a straightforward way of addressing structures located at different places in memory (see Figure 11-4). A base register can be pointed at the base of the structure, and elements of the structure can be addressed by their displacements from the base. Different copies of the same structure can be accessed by changing the base register.

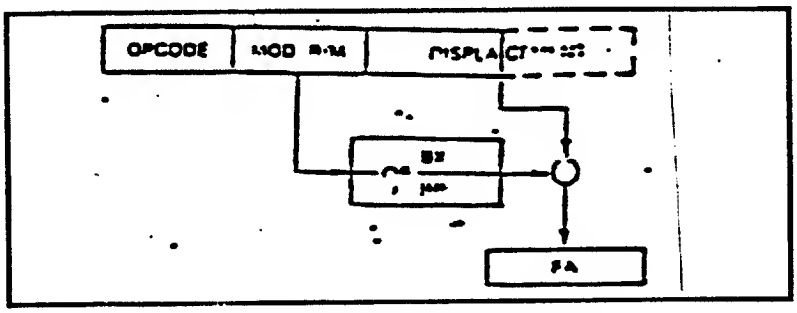

Figure 11-4. Based Addressing

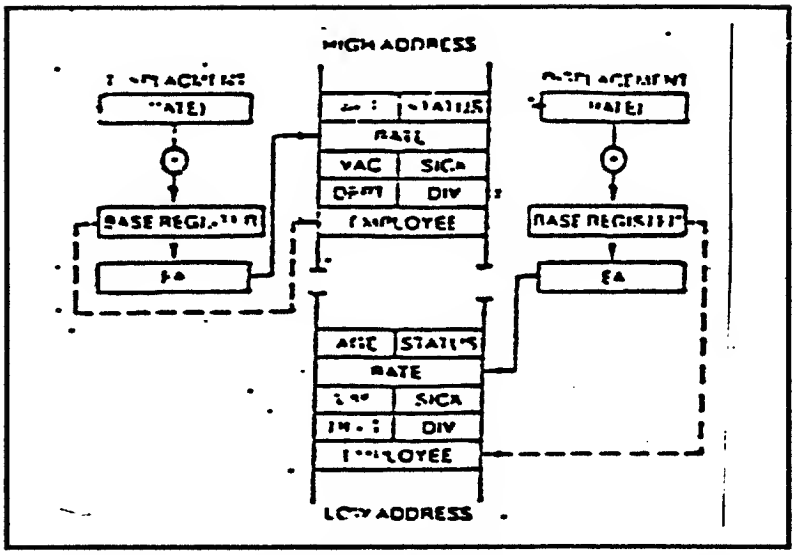

Figure 11-5. Accessing a Structure with Based Addressing

11.7 Indexed Addressing

In indexed addressing, the effective address is calculated by the sum of a displacement plus the content of an index register (SI or DI) as shown in Figure 11-6. Indexed addressing is often used to access elements in an array (see Figure 11-7). The displacement locates the beginning of the array, and the value of the index register selects one element (the first element is selected if the index register contains 0). All array elements are the same length, so simple arithmetic on the index register selects any element.

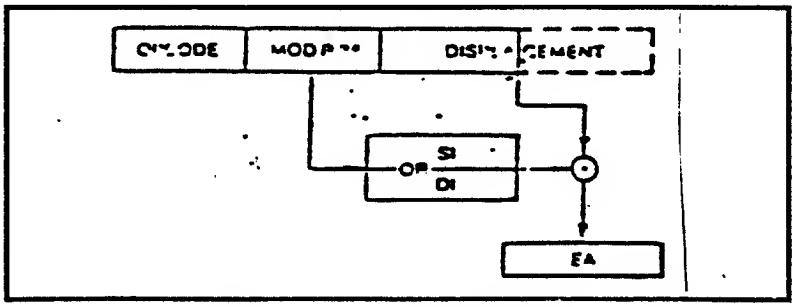

Figure 11-6. Indexed Addressing

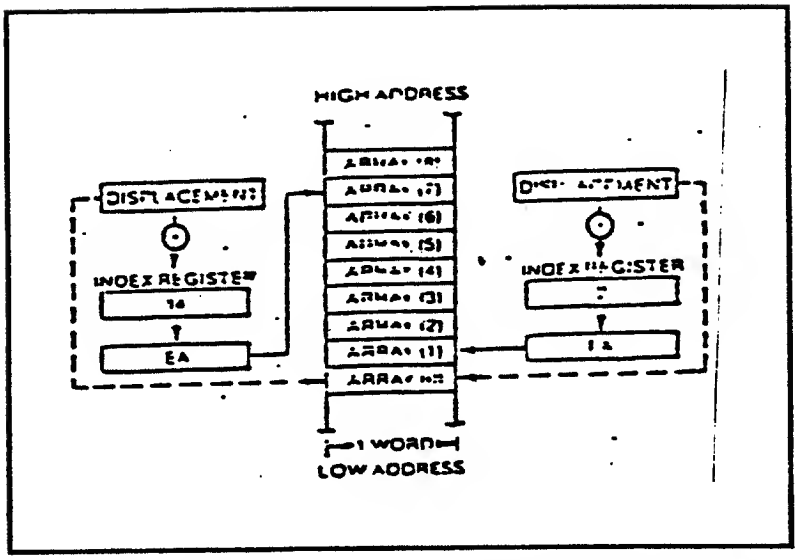

Figure 11-7. Accessing an Array with Indexed Addressing

11.8 Based Indexed Addressing

Based indexed addressing generates an effective address that is the sum of a base register, an index register, and a displacement (see Figure 11-8). Two address components can be varied at execution time, making based indexed addressing a very flexible mode. Based indexed addressing provides a convenient way for a procedure to address an array allocated on a stack (see Figure 11-9). Register BP can contain the offset of a reference point on the stack, typically the top of the stack after the procedure has saved registers and allocated local storage. The offset of the beginning of the array from the reference point can be expressed by a displacement value, and an index register can be used to access individual array elements.

Based indexed addressing can access arrays contained in structures and matrices (two-dimension arrays).

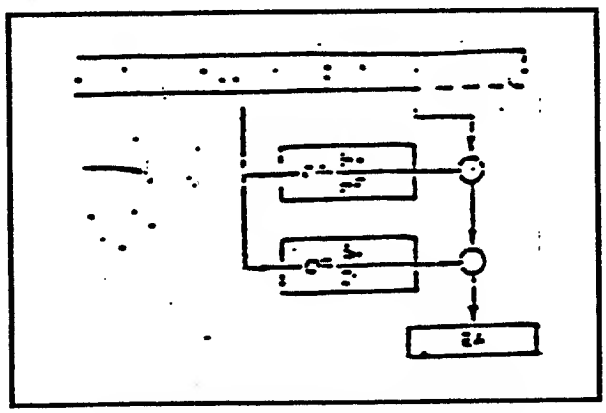

Figure 11-8. Based Indexed Addressing

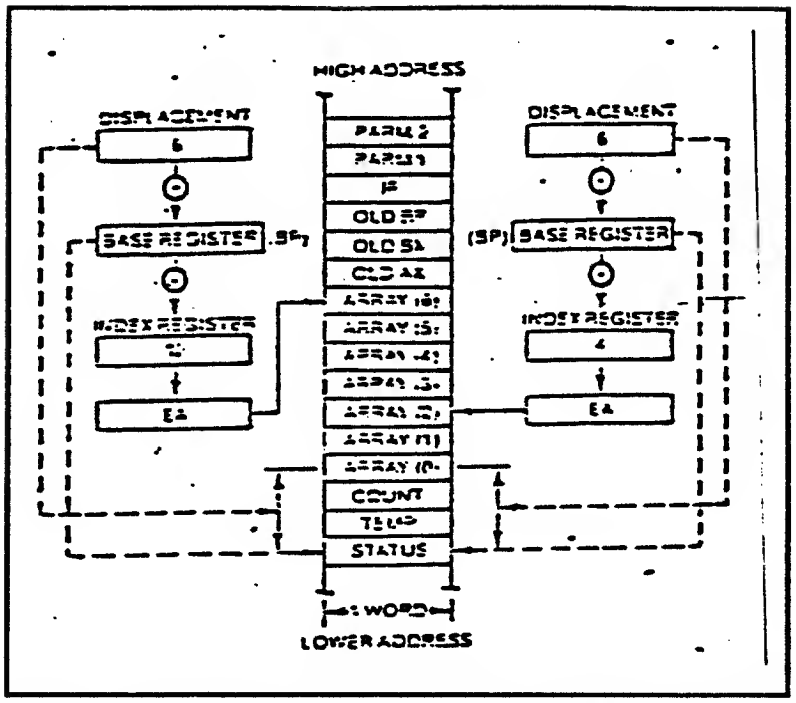

Figure 11-9. Addressing a Stack Array with Based Indexed Addressing

11.9 String Addressing

String instructions do not use the normal memory addressing modes to access their operands. Instead, the index registers are used implicitly as shown in Figure 11-10. When a string instruction is executed, SI is assumed to point to the first byte or word of the source string, and DI is assumed to point to the first byte or word of the destination string. In a repeated string operation, the CPUs automatically adjust SI and DI to obtain subsequent bytes or words.

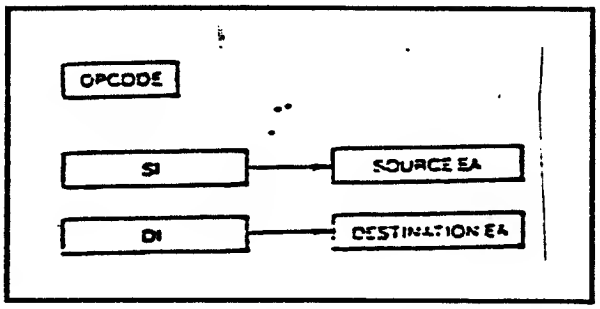

Figure 11-10. String Operand Addressing

11.10 I/O Port Addressing

When an I/O port is memory mapped, any of the memory operand addressing modes can be used to access the port. For example, a group of terminals can be accessed as an array. String instructions can also transfer data to memory-mapped ports with an appropriate hardware interface.

The two addressing modes that can be used to access ports located in the I/O space are illustrated in Figure 11-11. In direct port addressing, the port number is an 8-bit immediate operand. This allows fixed access to ports numbered 0-255. Indirect port addressing is similar to register indirect addressing of memory operands. The port number is taken from register DX and ranges from 0 to 65,535. By previously adjusting the content of register DX, one instruction can access any port in the I/O space. A group of adjacent ports can be accessed using a simple software loop that adjusts the value in DX.

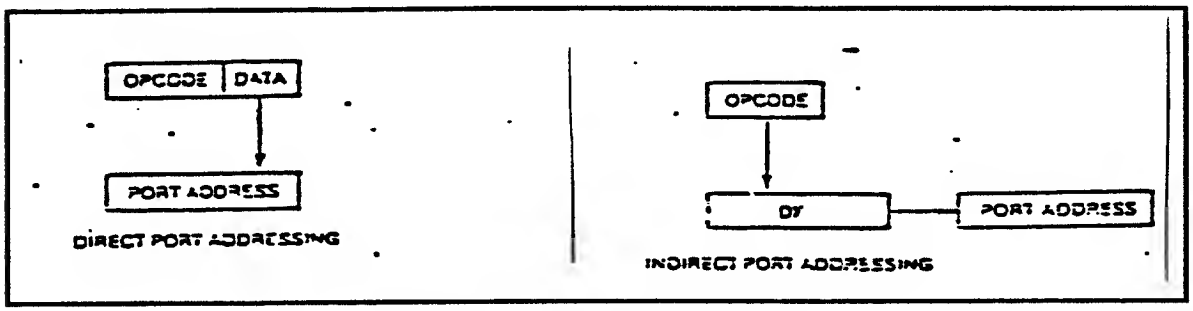

Figure 11-11. I/O Port Addressing

11.11 The 8088 Instruction Set

The 8086 and 8088 execute exactly the same instructions. This instruction set includes equivalents to the instruction typically found in previous microprocessors, such as the 8080/8085. Significant new operations include:

- Multiplication and division of signed and unsigned binary numbers as well as unpacked decimal numbers
- Move, scan, and compare operations for strings up to 64K bytes in length
- Non destructive bit testing
- Byte translation from one code to another
- Software generated interrupts
- A group of instructions that can help coordinate the activities of multiprocessor systems

These instructions treat different types of operands uniformly. Nearly every instruction can operate on either byte or word data. Register, memory, and immediate operands may be specified interchangeably in most instructions (except, of course, that immediate values may only serve as source and not destination operands). In particular, memory variables can be added to, subtracted from, shifted, compared, and so on, in place, without moving them in and out of registers. This saves instructions, registers, and execution time in assembly language programs. In high level languages, where most variables are memory based, compilers, such as PL/M-86, can produce faster and shorter object programs.

The 8086/8088 instruction set can be viewed as existing at two levels: the assembly level and the machine level. To the assembly language programmer, the 8086 and 8088 appear to have a repertoire of about 100 instructions. One MOV (move) instruction, for example, transfers a byte or a word from a register or a memory location or an immediate value to either a register or a memory location. The 8086 and 8088 CPUs, however, recognize 28 different MOV machine instructions ("move byte register to memory," "move word immediate to register," etc.). The ASM-86 assembler translates the assembly-level instructions written by a programmer into the machine level instructions that are actually executed by the 8086 or 8088. Compilers such as PL/M-86 translate high level language statements directly into machine level instructions.

The two levels of the instruction set address two different requirements: efficiency and simplicity. The numerous—there are about 300 in all—forms of machine level instructions allow these instructions to make very efficient use of storage. For example, the machine instruction that increments a memory operand is three or four bytes long because the address of the operand must be encoded in the instruction. To increment a register, however, does not require as much information, so the instruction can be shorter. In fact, the 8086 and 8088 have eight different machine level instructions that increment a different 16 bit register. These instructions are only one byte long.

If a programmer had to write one instruction to increment a register, another to increment a memory variable, etc., the benefit of compact instructions would be offset by the difficulty of programming. The assembly level instructions simplify the programmer's view of the instruction set. The programmer writes one form of the INC (increment) instruction and the ASM-86 assembler examines the operand to determine which machine level instruction to generate.

This section presents the 8088 instruction set from two perspectives. First, the assembly level instructions are described in functional terms. The assembly level instructions are then presented in a reference table that breaks out all permissible operand combinations with execution times and machine instruction length, plus the effect that the instruction has on the CPU flags.

11.12 Data Transfer Instructions

The 14 data transfer instructions (Table 11-1) move single bytes and words between memory and register as well as between register AL or AX and I/O ports. The stack manipulation instructions are included in this group as are instructions for transferring flag contents and for loading segment registers.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective adress
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

Table 11-1. Data Transfer Instructions

11.13 General Purpose Data Transfers

MOV *destination,source*

MOV transfers a byte or a word from the source operand to the destination operand.

PUSH *source*

PUSH decrements SP (the stack pointer) by two and then transfers a word from the source operand to the top of stack now pointed by SP. PUSH often is used to place parameters on the stack before calling a procedure. More generally, it is the basic means of storing temporary data on the stack.

POP *destination*

POP transfers the word at the current top of stack (pointed to by SP) to the destination operand, and then increments SP by two to point to the new top of stack. POP can be used to move temporary variables from the stack to registers or memory.

XCHG *destination, source*

XCHG (exchange) switches the contents of the source and destination (byte or word) operands. When used in conjunction with the LOCK prefix, XCHG can test and set a semaphore that controls access to a resource shared by multiple processors.

XLAT *translate-table*

XLAT (translate) replaces a byte in the AL register with a byte from a 256 byte, user coded translation table. Register BX is assumed to point to the beginning of the table. The byte in AL is used as an index into the table and is replaced by the byte at the offset in the table corresponding to AL's binary value. The first byte in the table has an offset of 0. For example, if AL contains 5_{16}, and the sixth element of the translation table contains 33_{16}, then AL will contain 33_{16} following the instruction. XLAT is useful for translating characters from one code to another, the classic example being ASCII to EBCDIC or the reverse.

IN *accumulator, port*

IN transfers a byte or a word, respectively, to the AL register or AX register, from an input port. The port number may be specified either with an immediate byte constant, allowing access to ports numbered 0 through 255, or with a number previously placed in the DX register, allowing variable access (by changing the value in DX) to ports numbered from 0 through 65,535.

OUT *port, accumulator*

OUT transfers a byte or a word from the AL register or the AX register, respectively, to an output port. The port number may be specified either with an immediate byte constant, allowing access to ports numbered 0 through 255, or with a number previously placed in register DDX, allowing variable access (by changing the value in DX) to ports numbered from 0 through 65,535).

11.14 Address Object Transfers

These instructions manipulate the addresses of variables rather than the contents or values of variables. They are most useful for list processing, based variables, and string operations.

LEA *destination, source*

LEA (load effective address) transfers the offset of the source operand (rather than its value) to the destination operand. The source operand must be a memory operand, and the destination operand must be a 16 bit general register. LEA does not affect any flags. The XLAT and string instructions assume that certain registers point to operands. LEA can be used to lead these registers (e.g., loading BX with the address of the translate table used by the XLAT instruction).

LDS *destination, source*

LDS (load pointer using DS) transfers a 32 bit pointer variable from source operand, which must be a memory operand, to the destination operand and register DS. The offset word of the pointer is transferred to the destination operand, which may be any 16 bit general register. The segment word of the pointer is transferred to register DS. Specifying SI as the destination operand is a convenient way to prepare to process a source string that is not in the current data segment (string instructions assume that the source string is located in the current data segment and that SI contains the offset of the string).

LES *destination, source*

LES (load pointer using ES) transfers a 32 bit pointer variable from the source operand, which must be a memory operand, to the destination operand and register ES. The offset word of the pointer is transferred to the destination operand, which may be any 16 bit general register. The segment word of the pointer is transferred to register ES. Specifying DI as the destination operand is a convenient way to prepare to process a destination string that is not in the current extra segment. (The destination string must be located in the extra segment, and DI must contain the offset of the string.)

11.15 Flag Transfers

LAHF

LAHF (load register AH from flags) copies SF, ZF, AF, PF and CF (the 8080/8085 flags) into bits 7, 6, 4, 2 and 0, respectively, of register AH (see Figure 11-1). The content of bits 5, 3 and 1 is undefined. The flags themselves are not affected. LAHF is provided primarily for converting 8080/8085 assembly language programs to run on an 8086 or 8088.

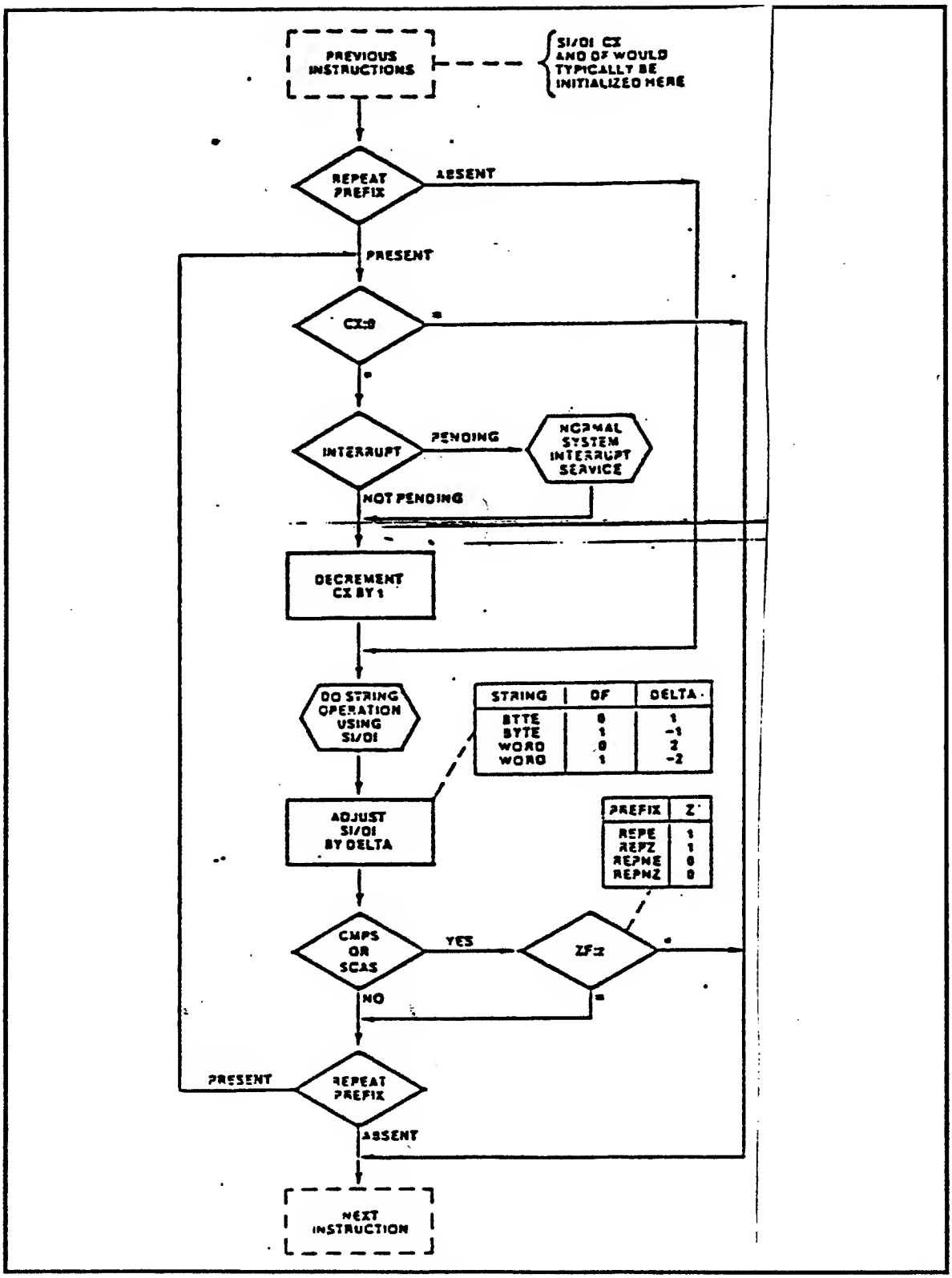

Figure 11-1. String Operation Flow

SAHF

SAHF (store register AH into flags) transfers bits 7, 6, 4, 2, and 0 from register AH into SF, ZF, AF, PF, and CF, respectively, replacing whatever values these flags previously had. OF, DF, IF and TF are not affected. This instruction is provided for 8080/8085 compatibility.

PUSHF

PUSHF decrements SP (the stack pointer) by two and then transfers all flags to the word at the top of stack pointed to be SP (see Figure 11-1). The flags themselves are not affected.

POPF

POPF transfers specific bits from the word at the current top of stack (pointed to by register SP) into the 8086/8088 flags, replacing whatever values the flags previously contained (Figure A-2. SP is then incremented by two to point to the new top of stack. PUSHF and POPF allow a procedure to save and restore a calling program's flags. They also allow a program to change the setting of TF (there is no instruction for updating this flag directly). The change is accomplished by pushing the flags, altering bit 8 of the memory image, and then popping the flags.

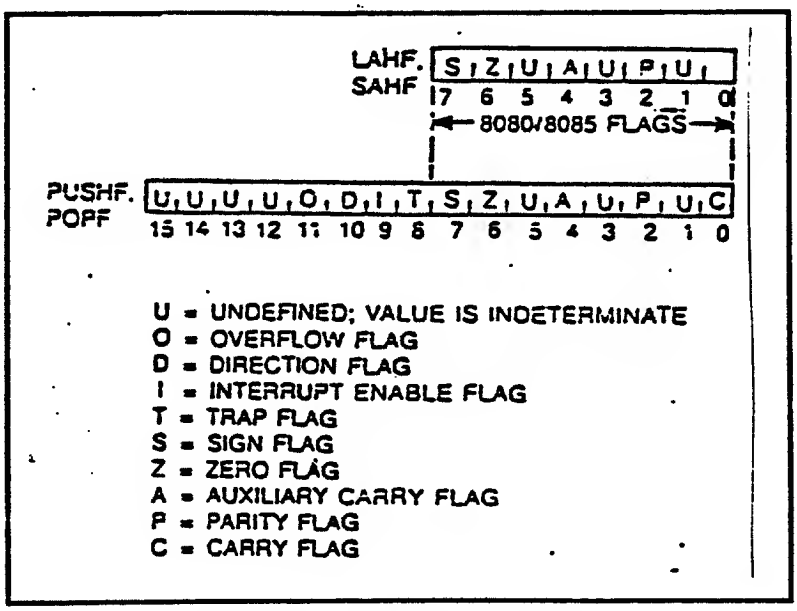

Figure 11-2. Flag Storage Formats

11.16 Arithmetic Instructions

11.16.1 Arithmetic Data Formats

8088 arithmetic operations (Table 11-2) may be performed on four types of numbers: unsigned binary, signed binary (integers), unsigned packed decimal and unsigned unpacked decimal (see Table 11-3). Binary numbers may be 8 or 16 bits long. Decimal numbers are stored in bytes, two digits per byte for packed decimal and one digit per byte for unpacked decimal. The processor always assumes that the operands specified in arithmetic instructions contain data that represent valid numbers for the type of instruction being performed. Invalid data may produce unpredictable results.

	ADDITION
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
	SUBTRACTION
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow.
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
	MULTIPLICATION
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
	DIVISION
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

Table 11-2. Arithmetic Instructions

HEX DECIMAL	UNSIGNED BIT	SIGNED PATTERN	UNPACKED BINARY	PACKED BINARY	DECIMAL
07	00000111	7	+7	7	7
89	10001001	137	−119	Invalid	89
C5	11000101	197	−59	Invalid	Invalid

Table 11-3. Arithmetic Interpretation of 8 Bit Numbers

Unsigned binary numbers may be either 8 or 16 bits long. All are considered in determining a number's magnitude. The value range of an 8 bit unsigned binary number is 0 through 255_{10}. Sixteen bits can represent values from 0 through $65{,}535_{10}$. Addition, subtraction, multiplication, and division operations are available for unsigned binary numbers.

Signed binary numbers (integers) may be either 8 or 16 bits long. The high order (leftmost) bit is interpreted as the number's sign: 0 = positive, and 1 = negative. Negative numbers are represented in standard two's complement notation. Since the high order bit is used for a sign, the range of an 8 bit integer is −128 through +127. The range of a 16 bit integer is from −32,768 through +32,767. The value zero has a positive sign. Multiplication and division operations are provided for signed binary numbers. Addition and subtraction are performed with the unsigned binary instructions. Conditional jump instructions, as well as an "interrupt on overflow" instruction, can be used following an unsigned operation on an integer to detect overflow into the sign bit.

Packed decimal numbers are stored as unsigned byte quantities. The byte is treated as having one decimal digit in each half byte (nibble). The digit in the high order half byte is the most significant. Hexadecimal values 0-9 are valid in each half byte, and the range of a packed decimal number is 0-99. Addition and subtraction are performed in two steps. First an unsigned binary instruction is used to produce an intermediate result in register AL. Then an adjustment operation is performed which changes the intermediate value in AL to a final correct packed decimal result. Multiplication and division adjustments are not available for packed decimal numbers.

Unpacked decimal numbers are stored as unsigned byte quantities. The magnitude of the number is determined from the low order half byte. Hexadecimal values 0-9 are valid and are interpreted as decimal numbers. The high order half byte must be zero for multiplication and division. It may contain any value for addition and subtraction. Arithmetic on unpacked decimal numbers is performed in two steps. The unsigned binary addition, subtraction, and multiplication operations are used to produce an intermediate result in register AL. An adjustment instruction then changes the value in AL to a final correct unpacked decimal number. Division is performed similarly, except that the adjustment is carried out on the numerator operand in register AL first, and then a following unsigned binary division instruction produces a correct result.

Unpacked decimal numbers are similar to the ASCII character representations of the digits 0-9. Note, however, that the high order half byte of an ASCII numeral is always 3_{16}. Unpacked decimal arithmetic may be performed on ASCII numeric characters under the following conditions:

- The high order half byte of an ASCII numeral must be set to 0_{16} prior to multiplication or division.
- Unpacked decimal arithmetic leaves the high order half byte set to 0_{16}. It must be set to 3_{16} to produce a valid ASCII numeral.

11.16.2 Flags

The 8086/8088 arithmetic instructions post certain characteristics of the result of the operation to six flags. Most of these flags can be tested by following the arithmetic instruction with a conditional jump instruction. The INTO (interrupt on overflow) instruction also may be used. The various instructions affect the flags differently, as explained in the instruction descriptions. However, they follow these general rules:

- CF (carry flag): If an addition results in a carry out of the high order bit of the result, then CF is set. Otherwise CF is cleared. If a subtraction results in a borrow into the high order bit of the result, then CF is set. Otherwise CF is cleared. Note that a signed carry is indicated by CF=OF. CF can be used to detect an unsigned overflow. Two instructions, ADC (add with carry) and SBB (subtract with borrow), incorporate the carry flag in their operations and can be used to perform multibyte (e.g., 32 bit, 64 bit) addition and subtraction.

- AF (auxiliary carry flag): If an addition results in a carry out of the low order half byte of the result, then AF is set. Otherwise AF is cleared. If a subtraction results in a borrow into the low order half byte of the result, then AF is set. Otherwise AF is cleared. The auxiliary carry flag is provided for the decimal adjust instructions and ordinarily is not used for any other purpose.

- SF (sign flag): Arithmetic and logical instructions set the sign flag equal to the high order bit (bit 7 or 15) of the result. For signed binary numbers, the sign flag will be 0 for positive results and 1 for negative results (so long as overflow does not occur). A conditional jump instruction can be used following addition or subtraction to alter the flow of the program depending on the sign of the result. Programs performing unsigned operations typically ignore SF since the high order bit of the result is interpreted as a digit rather than a sign.

- ZF (zero flag): If the result of an arithmetic or logical operation is zero, then ZF is set. Otherwise ZF is cleared. A conditional jump instruction can be used to alter the flow of the program if the result is or is not zero.

- PF (parity flag): If the low order eight bits of an arithmetic or logical result contain an even number of 1 bits, then the parity flag is set. Otherwise it is cleared.

- PF is provided for 8080/8085 compatibility. It also can be used to check ASCII characters for correct parity.

- OF (overflow flag): If the result of an operation is too large a positive number, or too small a negative number to fit in the destination operand (excluding the sign bit), then OF is set. Otherwise OF is cleared. OF thus indicates signed arithmetic overflow. It can be tested with a conditional jump or the INTO (interrupt on overflow) instruction. OF may be ignored when performing unsigned arithmetic.

11.16.3 Addition

ADD *destination, source*

The sum of the two operands, which may be bytes or words, replaces the destination operand. Both operands may be signed or unsigned binary numbers (see AAA and DAA). ADD updates AF, CF, OF, PF, SF, and ZF.

ADC *destination, source*

ADC (Add with Carry) sums the operands, which may be bytes or words, adds one if CF is set, and replaces the destination operand with the result. Both operands may be signed or unsigned binary numbers (see AAA and DAA). ADC updates AF, CF, OF, PF, SF, and ZF. Since ADC incorporates a carry from a previous operation, it can be used to write routines to add numbers longer than 16 bits.

INC *destination*

INC (Increment) adds one to the destination operand. The operand may be a byte or a word and is treated as an unsigned binary number (see AAA and DAA). INC updates AF, OF, PF, SF, and ZF. It does not affect CF.

AAA

AAA (ASCII Adjust for Addition) changes the contents of register AL to a valid unpacked decimal number. The high order half byte is zeroed. AAA updates AF and CF. The content of OF, PF, SF, and ZF is undefined following execution of AAA.

DAA

DAA (Decimal Adjust for Addition) corrects the result of previously adding two valid packed decimal operands (the destination operand must have been register AL). DAA changes the content of AL to a pair of valid packed decimal digits. It updates AF, CF, PF, SF, and ZF. The content of OF is undefined following execution of DAA.

11.16.4 Subtraction

SUB *destination, source*

The source operand is subtracted from the destination operand, and the result replaces the destination operand. The operands may be bytes or words. Both operands may be signed or unsigned binary numbers (see AAS and DAS). SUB updates AF, CF, OF, PF, SF, and ZF.

SBB *destination, source*

SBB (Subtract with Borrow) subtracts the source from the destination, subtracts one if CF is set, and returns the result to the destination operand. Both operands may be bytes or words. Both operands may be signed or unsigned binary numbers (see AAS and DAS). SBB updates AF, CF, OF, PF, SF, and ZF. Since it incorporates a borrow from a previous operation, SBB may be used to write routines that subtract numbers longer than 16 bits.

DEC *destination*

DEC (Decrement) subtracts one from the destination, which may be a byte or a word. DEC updates AF, OF, PF, SF, and ZF. It does not affect CF.

NEG *destination*

NEG (Negate) subtracts the destination operand, which may be a byte or a word, from 0 and returns the result to the destination. This forms the two's complement of the number, effectively reversing the sign of an interger. If the operand is zero, its sign is not changed. Attempting to negate a byte containing −128 or a word containing −32,768 causes no change to the operand and sets OF. NEG updates AF, CF, OF, PF, SF, and ZF. CF is always set except when the operand is zero, in which case it is cleared.

CMP *destination, source*

CMP (Compare) subtracts the source from the destination, which may be bytes or words, but does not return the result. The operands are unchanged, but the flags are updated and can be tested by a subsequent conditional jump instruction. CMP updates AF, CF, OF, PF, SF, and ZF. The comparison reflected in the flags is that of the destination to the source. If a CMP instruction is followed by a JG (Jump if Greater) instruction, for example, the jump is taken if the destination operand is greater than the source operand.

AAS

AAS (ASCII Adjust for Subtraction) corrects the result of a previous subtraction of two valid unpacked decimal operands (the destination operand must have been specified as register AL). AAS changes the content of AL to a valid unpacked decimal number. The high order half byte is zeroed. AAS updates AF and CF. The content of OF, PF, SF, and ZF is undefined following execution of AAS.

DAS

DAS (Decimal Adjust for Subtraction) corrects the result of a previous subtraction of two valid packed decimal operands (the destination operand must have been specified as register AL). DAS changes the content of AL to a pair of valid packed decimal digits. DAS updates AF, CF, PF, SF, and ZF. The content of OF is undefined following execution of DAS.

11.16.5 Multiplication

MUL *source*

MUL (Multiply) performs an unsigned multiplication of the source operand and the accumulator. If the source is a byte, then it is multiplied by register AL, and the double length result is returned in AH and AL. If the source operand is a word, then it is multiplied by register AX, and the double length result is returned in registers DX and AX. The operands are treated as unsigned binary numbers (see AAM). If the upper half of the result (AH for byte source, DX for word source) is nonzero, CF and OF are set. Otherwise they are cleared. When CF and OF are set, they indicate that AH or DX contains significant digits of the result. The content of AF, PF, SF, and ZF is undefined following execution of MUL.

IMUL *source*

IMUL (Integer Multiply) performs a signed multiplication of the source operand and the accumulator. If the source is a byte, then it is multiplied by register AL, and the double length result is returned in AH and AL. If the source is a word, then it is multiplied by register AX, and the double length result is returned in registers DX and AX. If the upper half of the result (AH for byte source, DX for word source) is not the sign extension of the lower half of result, CF and OF are set. Otherwise they are cleared. When CF and OF are set, they indicate that AH or DX contains significant digits of the result. The content of AF, PF, SF, and ZF is undefined following execution of IMUL.

AAM

AAM (ASCII Adjust for Multiply) corrects the result of a previous multiplication of two valid unpacked decimal operands. A valid 2 digit unpacked decimal number is derived from the content of AH and AL and is returned to AH and AL. The high order half bytes of the multiplied operands must have been 0_{16} for AAM to produce a correct result. AAM updates PF, SF, and ZF. The content of AF, CF, and OF is undefined following execution AAM.

11.16.6 Division

DIV *source*

DIV (divide) performs an unsigned division of accumulator (and its extension) by the source operand. If the source operand is a byte, it is divided into the double length dividend assumed to be in registers AL and AH. The single length quotient is returned in AL, and the single length remainder is returned in AH. If the source operand is a word, it is divided into the double length dividend in registers AX and DX. The single length quotient is returned in AX, and the single length remainder is returned in DX. If the quotient exceeds the capacity of its destination register (FF_{16} for byte source, $FFFF_{16}$ for word source), as when division by zero is attempted, a type 0 interrupt is generated, and the quotient and remainder are undefined. Nonintegral quotients are truncated to integers. The content of AF, CF, OF, PF, SF, and ZF is undefined following execution of DIV.

IDIV *source*

IDIV (Integer Divide) performs a signed division of the accumulator (and its extension) by the source operand. If the source operand is a byte, it is divided into the double length dividend assumed to be in registers AL and AH. The single length quotient is returned in AL, and the single length remainder is returned in AH. For byte integer division, the maximum positive quotient is $+127(7F_{16})$ and the minimum negative quotient is $-127(81_{16})$. If the source operand is a word,it is divided into the double length dividend in registers AX and DX. The single length quotient is returned in AX, and the single length remainder is returned in DX. For word integer division, the maximum positive quotient is +32,767 ($7FFF_{16}$) and the minimum negative quotient is −32,767 (8001_{16}). If the quotient is positive and exceeds the maximum, or is negative and is less than the minimum, the quotient and remainder are undefined, and a type 0 interrupt is generated. In particular, this occurs if division by 0 is attempted. Nonintegral quotients are truncated (toward 0) to integers, and the remainder has the same sign as the dividend. The content of AF, CF, OF, PF, SF, and ZF is undefined following IDIV.

AAD

AAD (ASCII Adjust for Division) modifies the numerator in AL before dividing two valid unpacked decimal operands so that the quotient produced by the division will be a valid unpacked decimal number. AH must be zero for the subsequent DIV to produce the correct result. The quotient is returned in AL, and the remainder is returned in AH. Both high order half bytes are zeroed. AAD updates PF, SF, and ZF. The content of AF, CF, and OF is undefined following execution of AAD.

CBW

CBW (Convert Byte to Word) extends the sign of the byte in register AL throughout register AH. CBW does not affect any flags. CBW can be used to produce a double length (word) dividend from a byte prior to performing byte division.

CWD

CWD (Convert Word to Doubleword) extends the sign of the word in register DX. CWD does not affect any flags. CWD can be used to produce a double length (doubleword) dividend from a word prior to performing word division.

11.17 Bit Manipulation Instructions

The 8086 and 8088 provide three groups of instructions (Table A-4) for manipulating bits within both bytes and words: logical, shifts, and rotates.

	LOGICALS
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
	SHIFTS
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
	ROTATES
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

Table 11-4. Bit Manipulation Instructions

11.17.1 Logical

The logical instructions include the boolean operators "not", "and", "inclusive or", and "exclusive or", plus a TEST instruction that sets the flags, but does not alter either of its operands.

AND, OR, XOR and TEST affect the flags as follows: The overflow (OF) and carry (CF) flags are always cleared by logical instructions, and the content of the auxiliary carry (AF) flag is always undefined following execution of a logical instruction. The sign (SF), zero (ZF) and parity (PF) flags are always posted to reflect the result of the operation and can be tested by conditional jump instructions. The interpretation of these flags is the same as for arithmetic instructions. SF is set if the result is negative (high order bit is 1), and is cleared if the result is positive (high order bit is 0). ZF is set if the result is zero. It is otherwise cleared. PF is set if the result contains an even number of 1 bits (has even parity) and is cleared if the number of 1 bits is odd (the result has odd parity). Note that NOT has no effect on the flags.

NOT *destination*

NOT inverts the bits (forms the one's complement) of the byte or word operand.

AND *destination, source*

AND performs the logical "and" of the two operands (byte or word) and returns the result to the destination operand. A bit in the result is set if both correspondence bits of the original operands are set. Otherwise the bit is cleared.

OR *destination, source*

OR performs the logical "inclusive or" of the two operands (byte or word) and returns the result to the destination operand. A bit in the result is set if either or both corresponding bits in the original operands are set. Otherwise the result bit is cleared.

XOR *destination, source*

XOR (Exclusive Or) performs the logical "exclusive or" of the two operands and returns the result to the destination operand. A bit in the result is set if the corresponding bits of the original operands contain opposite values (one is set, the other is cleared). Otherwise the result bit is cleared.

TEST *destination, source*

TEST performs the logical "and" of the two operands (byte or word), updates the flags, but does not return the result—i.e., neither operand is changed. If a TEST instruction is followed by a JNZ (Jump if Not Zero) instruction, the jump will be taken if there are any corresponding 1 bits in both operands.

11.17.2 Shifts

The bits in bytes and words may be shifted arithmetically or logically. Up to 255 shifts may be performed, according to the value of the count operand coded in the instruction. The count may be specified as the constant 1, or as register CL, allowing the shift count to be a variable supplied at execution time. Arithmetic shifts may be used to multiply and divide binary numbers by powers of two (see note in description of SAR). Logical shifts can be used to isolate bits in bytes or words.

Shift instructions affect the flags as follows: AF is always undefined following a shift operation. PF, SF, and ZF are updated normally, as in the logical instructions. CF always contains the value of the last bit shifted out of the destination operand. The content of OF is always undefined following a multibit shift. In a single bit shift, OF is set if the value of the high order (sign) bit was changed by the operation. If the sign bit retains its original value, OF is cleared.

SHL/SAL*destination, count*

SHL and SAL (Shift Logical Left and Shift Arithmetic Left) perform the same operation and are physically the same instruction. The destination byte or word is shifted left by the number of bits specified in the count operand. Zeros are shifted in on the right. If the sign bit retains its original value, then IF is cleared.

SHR *destination, source*

SHR (Shift Logical Right) shifts the bits in the destination operand (byte or word) to the right by the number of bits specified in the count operand. Zeros are shifted in on the left. If the sign bit retains its original value, then OF is cleared.

SAR *destination, count*

SAR (Shift Arithmetic Right) shifts the bits in the destination operand (byte or word) to the right by the number of bits specified in the count operand. Bits equal to the original high order (sign) bit are shifted in on the left, preserving the sign of the original value. Note that SAR does not produce the

same result as the dividend of an equivalent IDIV instruction if the destination operand is negative and 1 bits are shifted out. For example, shifting −5 right by one bit yields −3, while integer division of −5 by 2 yields −2. The difference in the instructions is that IDIV truncates all numbers toward zero, while SAR truncates positive numbers toward zero and negative numbers toward negative infinity.

11.17.3 Rotates

Bits in bytes and words also may be rotated. Bits rotated out of an operand are not lost as in a shift, but are circled back into the other end of the operand. As in the shift instructions, the number of bits to be rotated is taken from the count operand, which may specify either a constant of l, or the CL register. The carry flag may act as an extension of the operand in two of the rotate instructions, allowing a bit to be isolated in CF and then tested by a JC (Jump if Carry) or JNC (Jump if Not Carry) instruction.

Rotates affect only the carry and overflow flags. CF always contains the value of the last bit rotated out. On multibit rotates, the value of OF is always undefined. In single bit rotates, OF is set if the operation changes the high order (sign) bit of the destination operand. If the sign bit retains its original value, OF is cleared.

ROL *destination, count*

ROL (Rotate Left) rotates the destination byte or word left by the number of bits specified in the count operand.

ROL (Rotate Right) operates similar to ROL except that the bits in the destination byte or word are rotated right instead of left.

RCL *destination, count*

RCL (Rotate through Carry Left) rotates the bits in the byte or word destination operand to the left by the number of bits specified in the count operand. The carry flag (CF) is treated as "part of" the destination operand. That is, its value is rotated into the low order bit of the destination, and is itself replaced by the high order bit of the destination.

RCR *destination, count*

RCR (Rotate through Carry Right) operates exactly like RCL except that the bits are rotated right instead of left.

11.18 String Instructions

Five basic string operations, called primitives, allow strings of bytes or words to be operated on, one element (byte or word) at a time. Strings of up to 64k bytes may be manipulated with these instructions. Instructions are available to move, compare, and scan for a value, as well as for moving string elements to and from the accumulator (see Table 11-5). These basic operations may be preceded by a special one byte prefix that causes the instruction to be repeated by the hardware, allowing long strings to be processed much faster than would be possible with a software loop. The repetitions can be terminated by a variety of conditions, and a repeated operation may be interrupted and resumed.

STRING INSTRUCTIONS

REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
MOVS	Move byte or word string
MOVSB/MOVSW	Move byte or word string
CMPS	Compare byte or word string
SCAS	Scsan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string

Table 11-5. String Instructions

The string instructions operate quite similarly in many respects. The common characteristics are covered here and in Table 11-6 and Figure 11-2 rather than in the descriptions of the individual instructions. A string instruction may have a source operand, a destination operand, or both. The hardware assumes that a source string resides in the current data segment. A segment prefix byte may be used to override this assumption. A destination string must be in the current extra segment. The assembler checks the attributes of the operands to determine if the elements of the strings are bytes or words. The assembler does not, however, use the operand names to address the strings. Rather, the content of register SI (source index) is used as an offset to address the current element of the source string, and the content of register DI (destination index) is taken as the offset of the current destination string element. These registers must be initialized to point to the source/destination strings before executing the string instruction. The LDS, LES, and LEA instructions are useful in this regard.

STRING INSTRUCTION REGISTER AND FLAG USE

SI	Index (offset) for source string
DI	Index (offset) for destination
CX	Repetition counter
AL/AX	Scan value Destination for LODS Source for STOS
DF	0 = auto-increment SI, DI 1 = auto-decrement SI, DI
ZF	Scan/compare terminator

Table 11-6. String Instruction Register and Flag Use

The string instructions automatically update SI and/or DI in anticipation of processing the next string element. The DF (direction flag) setting determines whether the index registers are auto decremented (DF = 1). If byte strings are being processed, SI and/or DI is adjusted by 1. The adjustment is 2 for word strings.

If a Repeat prefix has been coded, then register CX (count register) is decremented by 1 after each repetition of the string instruction. Therefore, CX must be initialized to the number of repetitions desired before the string instruction is executed. If CX is 0, the string instruction is not executed, and control goes to the following instruction.

Section 2.10 contains examples that illustrate the use of all the string instructions.

11.19 Repeat Prefixes

REP (Repeat), REPE (Repeat While Equal), REPZ (Repeat While Zero), REPNE (Repeat While Not Equal), and REPNZ (Repeat While Not Zero) are five mnemonics for two forms of the prefix byte that controls repetition of a subsequent string instruction. The different mnemonics are provided to improve program clarity. The repeat prefixes do not affect the flags.

REP is used in conjunction with the MOVS (Move String) and STOS (Store String) instructions and is interpreted as "repeat while not end-of-string" (CX not 0). REPE and REPZ operate identically and are physically the same prefix byte as REP. These instructions are used with the CMPS (Compare String) and SCAS (Scan String) instructions and require ZF (posted by these instructions) to be set before initiating the next repetition. REPNE and REPNZ are two mnemonics for the same prefix byte. These instructions function the same as REPE and REPZ, except that the zero flag must be cleared or the repetition is terminated. Note that ZF does not need to be initialized before executing the repeated string instruction.

Repeated string sequences are interruptable. The processor will recognize the interrupt before processing the next string element. System interrupt processing is not affected in any way. Upon return from the interrupt, the repeated operation is resumed from the point of interruption. Note, however, that execution does not resume properly if a second or third prefix (i.e., segment override or LOCK) has been specified in addition to any of the repeat prefixes. The processor "remembers" only one prefix in effect at the time of the interrupt–the prefix that immediately precedes the string instruction. After returning from the interrupt, processing resumes at this point, but any additional prefixes specified are not in effect. If more than one prefix must be used with a string instruction, interrupts may be disabled for the duration of the repeated execution. However, this will not prevent a nonmaskable interrupt from being recognized. Also, the time that the system is unable to respond to interrupts may be unacceptable if long strings are being processed.

MOVS *destination-string, source-string*

MOVS (Move String) transfers a byte or a word from the source string (addressed by SI) to the destination string (addressed by DI) and updates SI and DI to point to the next string element. When used in conjunction with REP, MOVS performs a memory-to-memory block transfer.

MOVSB

MOVSW

MOVSB and MOVSW are alternate mnemonics for the move string instruction. These mnemonics are coded without operands. They explicitly tell the assembler that a byte string (MOVSB) or a word string (MOVSW) is to be moved (when MOVS is coded, the assembler determines the string type from

the attributes of the operands). These mnemonics are useful when the assembler cannot determine the attributes of a string—e.g., when a section of code is being moved.

CMPS *destination-string, source-string*

CMPS (Compare String) subtracts the destination byte or word (addressed by DI) from the source byte or word (addressed by SI). CMPS affects flags without altering either operand, updates SI and DI to point to the next string element, and updates AF, CF, OF, PF, SF, and ZF to reflect the relationship of the destination element to the source element. For example, if a JG (Jump if Greater) instruction follows CMPS, the jump is taken if the destination element is greater than the source element. If CMPS is prefixed with REPE or REPZ, the operation is interpreted as "compare while not end-of-string (CX not zero) and strings are equal (ZF = 1)." If CMPS is preceded by REPNE or REPNZ, the operation is interpreted as "compare while not end-of-string (CX not zero) and strings are not equal (ZF = 0)." Thus, CMPS can be used to find matching or differing string elements.

SCAS *destination-string*

SCAS (Scan String) subtracts the destination string element (byte or word) addressed by DI from the content of AL (byte string) or AX (word string) and updates the flags, but does not alter the destination string or the accumulator. SCAS also updates DI to point to the next string element and AF, CF, OF, PF, SF, and ZF to reflect the relationship of the scan value in AL/AX to the string element. If SCAS is prefixed with REPE or REPZ, the operation is interpreted as "scan while not end-of-string (CX not 0) and string-element = scan value (ZF = 1)." This form may be used to scan for departure from a given value. If SCAS is prefixed with REPNE or REPNZ, the operation is interpreted as "scan while not end-of-string (CX not 0) and string-element is not equal to scan value (ZF = 0)." This form may be used to locate a value in a string.

LODS *source-string*

LODS (Load String) transfers the byte or word string element addressed by SI to register AL or AX, and updates SI to point to the next element in the string. This instruction is not ordinarily repeated since the accumulator would be overwritten by each repetition, and only the last element would be retained. However, LODS is very useful in software loops as part of a more complex string function built up from string primitives and other instructions.

STOS *destination-string*

STOS (Store String) transfers a byte or word from register AL or AX to the string element addressed by DI and updates DI to point to the next location in the string. As a repeated operation, STOS provides a convenient way to initialize a string to a constant value (e.g., to blank out a print line).

11.20 Program Transfer Instructions

The sequence of execution of instructions in an 8086/8088 program is determined by the content of the code segment register (CS) and the instruction pointer (IP). The CS register contains the base address of the current code segment, the 64k portion of memory from which instructions are presently being fetched. The IP is used as an offset from the beginning of the code segment. The combination of CS and IP points to the memory location from which the next instruction is to be fetched. (Recall that under most operating conditions, the next instruction to be executed has already been fetched from memory and is waiting in the CPU instruction queue.) The program transfer instructions operate on the instruction pointer and on the CS register. Changing the content of these causes normal sequential execution to be altered. When a program transfer occurs, the queue no longer contains the correct instruction, and the BIU obtains the next instruction from memory using the new IP and CS values, passes the instruction directly to the EU, and then begins refilling the queue from the new location.

Four groups of program transfers are available in the 8088: unconditional transfers, conditional transfers, iteration control instructions and interrupt related instructions (see Table 11-7). Only the interrupt related instructions affect any CPU flags. As will be seen, however, the execution of many of the program transfer instructions is affected by the states of the flags.

UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX=0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Table 11-7. Program Transfer Instructions

11.20.1 Unconditional Transfers

The unconditional transfer instructions may transfer control to a target instruction within the current code segment (intrasegment transfer) or to a different code segment (intersegment transfer). The ASM-86 assembler terms an intrasegment target NEAR and an intersegment target FAR.) The transfer is made unconditionally any time the instruction is executed.

CALL *procedure-name*

CALL activates an out-of-line procedure, saving information on the stack to permit a RET (return) instruction in the procedure to transfer control back to the instruction following the CALL. The

assembler generates one of two types of CALL instruction. The type depends on whether the programmer has defined the procedure name as NEAR or FAR. For control to return properly, the type of CALL instruction must match the type of RET instruction that exits from the procedure. (The potential for a mismatch exists if the procedure and the CALL are contained in separately assembled programs.) Different forms of the CALL instruction allow the address of the target procedure to be obtained from the instruction itself (direct CALL) or from a memory location or register referenced by the instruction (indirect CALL). In the following descriptions, bear in mind that the processor automatically adjusts IP to point to the next instruction to be executed before saving it on the stack.

For an intrasegment direct CALL, SP (the stack pointer) is decremented by two and IP is pushed onto the stack. The relative displacement (up to +32k) of the target procedure from the CALL instruction is then added to the instruction pointer. This form of the CALL instruction is self relative and is appropriate for position independent (dynamically relocatable) routines in which the CALL and its target are in the same segment and are moved together.

An intrasegment indirect CALL may be made through memory or through a register. SP is decremented by two and IP is pushed onto the stack. The offset of the target procedure is obtained from the memory word or 16 bit general register referenced in the instruction and replaces IP.

For an intersegment direct CALL, SP is decremented by two, and CS is pushed onto the stack. CS is replaced by the segment word contained in the instruction. SP again is decremented by two. IP is pushed onto the stack and is replaced by the offset word contained in the instruction.

For an intersegment indirect CALL (which only may be made through memory), SP is decremented by two, and CS is pushed onto the stack. CS is then replaced by the content of the second word of the doubleword memory pointer referenced by the instruction. SP again is decremented by two, and IP is pushed onto the stack and is replaced by the content of the first word of the doubleword pointer referenced by the instruction.

RET *optional-pop-value*

RET (Return) transfers control from a procedure back to the instruction following the CALL that activated the procedure. The assembler generates either an intrasegment RET, if the programmer has defined the procedure NEAR, or an intersegment RET, if the procedure has been defined as FAR. RET pops the word at the top of the stack (pointed to by register SP) into the instruction pointer and increments SP by two. If RET is intersegment, the word at the new top of stack is popped into the CS register, and SP is again incremented by two. If an optional pop value has been specified, RET adds that value to SP. This feature may be used to discard parameters pushed onto the stack before the execution of the CALL instruction.

JMP *Target*

JMP unconditionally transfers control to the target location. Unlike a CALL instruction, JMP does not save any information on the stack, and no return to the instruction following the JMP is expected. Like CALL, the address of the target operand may be obtained from the instruction itself (direct JMP) or from memory or a register referenced by the instruction (indirect JMP).

An intrasegment direct JMP changes the instruction pointer by adding the relative displacement of the target from the JMP instruction. If the assembler can determine that the target is within 127 bytes of the JMP, it automatically generates a two byte form of this instruction called a SHORT JMP. Otherwise, it generates a NEAR JMP that can address a target within +32k. Intrasegment direct JMPS are self relative and are appropriate in position independent (dynamically relocatable) routines in which the JMP and its target are in the same segment and are moved together.

An intrasegment indirect JMP may be made either through memory or through a 16 bit general register. In the first case, the content of the word referenced by the instruction replaces the instruction pointer. In the second case, the new IP value is taken from the register named in the instruction.

An intersegment direct JMP replaces IP and CS with values contained in the instruction.

An intersegment indirect JMP may be made only through memory. The first word of the doubleword pointer referenced by the instruction replaces IP, and the second word replaces CS.

11.20.2 Conditional Transfers

The conditional transfer instructions are jumps that may or may not transfer control depending on the state of the CPU flags at the time the instruction is executed. These 18 instructions (see Table 11-8) each test a different combination of flags for a condition. If the condition is true, then control is transferred to the target specified in the instruction. If the condition is false, then control passes to the instruction that follows the conditional jump. All conditional jumps are SHORT, that is, the target must be in the current code segment and within −128 to +127 bytes of the first byte of the next instruction (JMP 00_{16} jumps to the first byte of the next instruction). Since the jump is made by adding the relative displacement of the target to the instruction pointer, all conditional jumps are self relative and are appropriate for position independent routines.

MNEMONIC	CONDITION TESTED	"JUMP IF..."
JA/JNBE	(CF or ZF)=0	above/not below nor equal
JAE/JNB	CF=0	above or equal/not below
JB/JNAE	CF=1	below/not above nor equal
JBE/JNA	(CF or ZF)=1	below or equal/not above
JC	CF=1	carry
JE/JZ	ZF=1	equal/zero
JG/JNLE	((SF xor OF)or ZF)=0	greater/not less nor equal
JGE/JNL	(SF xor OF)=0	greater or equal/not less
JL/JNGE	(SF xor OF)=1	less/notgreater nor equal
JLE/JNG	((SF xor OF)or ZF)=1	less or equal/not greater
JNC	CF=0	not carry
JNE/JNZ	ZF=0	not equal/not zero
JNO	OF=0	not overflow
JNP/JPO	PF=0	not parity/parity odd
JNS	SF=0	not sign
JO	OF=1	overflow
JP/JPE	PF=1	parity/parity equal
JS	SF=1	sign

Table 11-8. Interpretation of Conditional Transfers

NOTE: "above" and "below" refer to the relationship of two unsigned values. "greater" and "less" refer to the relationship of two signed values.

11.20.3 Iteration Control

The iteration control instructions can be used to regulate the repetition of software loops. These instructions use the CX register as a counter. Like the conditional transfers, the iteration control instructions are self relative and may only transfer to targets that are within −128 to +127 bytes of themselves, i.e., they are SHORT transfers.

LOOP *short-label*

LOOP decrements CX by 1 and transfers control to the target operand if CX is not 0. Otherwise the instruction following LOOP is executed.

LOOPE *short-label*
LOOPZ *short-label*

LOOPE and LOOPZ (Loop While Equal and Loop While Zero) are different mnemonics for the same instruction (similar to the REPE and REPZ repeat prefixes). CX is decremented by 1, and control is transferred to the target operand if CX is not 0 and if ZF is set. Otherwise the instruction following LOOPE or LOOPZ is executed.

LOOPNE *short-label*
LOOPNZ *short-label*

LOOPNE and LOOPNZ (Loop While Not Equal and Loop While Not Zero) are also synonyms for the same instruction. CX is decremented by 1, and control is transferred to the target operand if CX is not 0 and ZF is clear. Otherwise the next sequential instruction is executed.

JCXZ *short-label*

JCXZ (Jump If CX Zero) transfers control to the target operand if CX is 0. This instruction is useful at the beginning of a loop to bypass the loop if CX has a zero value, i.e., to execute the loop zero times.

11.20.4 Interrupt Instructions

The interrupt instructions allow interrupt service routines to be activated by programs as well as by external hardware devices. The effect of software interrupts is similar to hardware initiated interrupts. However, the processor does not execute an interrupt acknowledge bus cycle if the interrupt originates in software or with an NMI. The effect of the interrupt instructions on the flags is covered in the description of each instruction.

INT *interrupt-type*

INT (Interrupt) activates the interrupt procedure specified by the interrupt-type operand. INT decrements the stack pointer by two, pushes the flags onto the stack, and clears the trap flag (TF) and interrupt enable flag (IF) to disable single-step and maskable interrupts. The flags are stored in the format used by the PUSHF instruction. SP is decremented again by two, and the CS register is pushed onto the stack. The address of the interrupt pointer is calculated by multiplying interrupt-type by four. The second word on the interrupt pointer replaces CS. SP again is decremented by two, and IP is pushed onto the stack and is replaced by the first word of the interrupt pointer. If interrupt-type = 3, the assembler generates a short (1 byte) form of the instruction, known as the breakpoint interrupt.

Software interrupts can be used as supervisor calls—requests for service from an operating system. A different interrupt-type can be used for each type of service that the operating system could supply

for an application program. Software interrupts also may be used to check out interrupt service procedures written for hardware initiated interrupts.

INTO

INTO (Interrupt on Overflow) generates a software interrupt if the overflow flag (OF) is set. Otherwise control proceeds to the following instruction without activating an interrupt procedure. INTO addresses the target interrupt pointer at location 10_{16}. It clears the TF and IF flags and otherwise operates like INT. INTO may be written following an arithmetic or logical operation to activate an interrupt procedure if overflow occurs.

IRET

IRET (Interrupt Return) transfers control back to the point of interruption by popping IP, CS, and the flags from the stack. IRET thus affects all flags by restoring them to previously saved values. IRET is used to exit any interrupt procedure, whether activated by hardware or software.

11.21 Processor Control Instructions

These instructions (see Table 11-9) allow programs to control various CPU functions. One group of instructions updates flags, and another group is used primarily for synchronizing the 8086 or 8088 with external events. A final instruction causes the CPU to do nothing. Except for the flag operations, none of the processor control instructions affect the flags.

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for TEST pin active
ESC	Escape to external processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation

Table 11-9. Processor Control Instructions

11.22 Flag Operations

CLC

CLC (Clear Carry flag) zeroes the carry flag (CF) and affects no other flags. It (and CMC and STC) is useful in conjunction with the RCL and RCR instructions.

CMC

CMC (Complement Carry flag) toggles CF to its opposite state and affects no other flags.

STC

STC (Set Carry flag) sets CF to 1 and affects no other flags.

CLD

CLD (Clear Direction flag) zeroes DF, causing the string instructions to auto-increment the SI and/or DI index registers. CLD does not affect any other flags.

STD

STD (Set Direction flag) sets DF to 1, causing the string instructions to autodecrement the SI and/or DI index registers. STD does not affect any other flags.

CLI

CLI (Clear Interrupt-enable flag) zeroes IF. When the interrupt- enable flag is cleared, the 8086 and 8088 do not recognize an external interrupt request that appears on the INTR line. In other words, maskable interrupts are disabled. A nonmaskable interrupt appearing on the NMI line, however, is honored, as is a software interrupt. CLI does not affect any other flags.

STI

STI (Set Interrupt-enable flag) sets IF to 1, enabling processor recognition of maskable interrupt requests appearing on the INTR line. Note however, that a pending interrupt will not actually be recognized until the instruction following STI has executed. STI does not affect any other flags.

11.23 External Synchronization

HLT

HLT (Halt) causes the 8088 to enter the halt state. The processor leaves the halt state upon activation of the RESET line, upon receipt of a nonmaskable interrupt request on NMI or, if interrupts are enabled, upon receipt of a maskable interrupt request on INTR. HLT does not affect any flags. It may be used as an alternative to an endless software loop in situations where a program must wait for an interrupt.

WAIT

WAIT causes the CPU to enter the wait state while its TEST line is not active. WAIT does not affect any flags.

ESC *external-opcode,source*

ESC (Escape) provides a means for an external processor to obtain an opcode and possibly a memory operand from the 8088. The external opcode is a 6 bit immediate constant that the assembler encodes in the machine instruction it builds (see Table A-10). An external processor may monitor the system bus and capture this opcode when the ESC is fetched. If the source operand is a register, the processor does nothing. If the source operand is a memory variable, the processor obtains the operand from memory and discards it. An external processor may capture the memory operand when the processor reads it from memory.

LOCK

LOCK is a 1 byte prefix that causes the 8088 (configured in maximum mode) to assert its bus LOCK signal while the following instruction executes. LOCK does not affect any flags. See section 2.5 for more information on LOCK.

NOP

NOP (No Operation) causes the CPU to do nothing. NOP does not affect any flags.

11.24 Instruction Set Reference Information

A subsequent chapter provides detailed operational information for the 8088 instruction set.

13. Miscellaneous Supply

13.1 Power Supply

The power supply for the VICTOR 9000 is designed for operational and equipment safety, single-switch operation, and data protection.

The power supply is a 4 voltage regulator with one +5V output, two +12V outputs, and one −12V output. Overall feedback regulates all outputs by sensing the +5V. The −12V output and one of the +12V outputs have independent series regulators.

The power supply provides 6 amps of +5V ±2%, 2 amps of +12V ±5%, 1.5 amps of +12V ±5%, and .2 amp of −12V ±5%. The operating range is 90-137V ac or 190-270V ac. The range may be selected and strapped by jumper wire. The power supply operates at 47-63 Hz. All power levels are regulated with overvoltage and overcurrent protection.

Line filters provide noise/ripple suppression and conducted or radiated radio frequency energy reduction.

When the power supply is shorted or overloaded, fold-back limiting occurs, preventing overheating. The unit withstands shorted output for an indefinite period and transients of up to 6000V peak. The power supply absorbs transients without causing any deviation at the output.

As shown in Figure 13-1, the power supply is in a shielded case, housed in the rear of the processor unit. The power supply module contains a fuse, a power switch and a line filter connector which connects to the AC power mains. It powers the processor unit, installed options, the display unit, and the keyboard unit. A 4" fan, mounted in the right rear of the processor unit, provides cooling air flow.

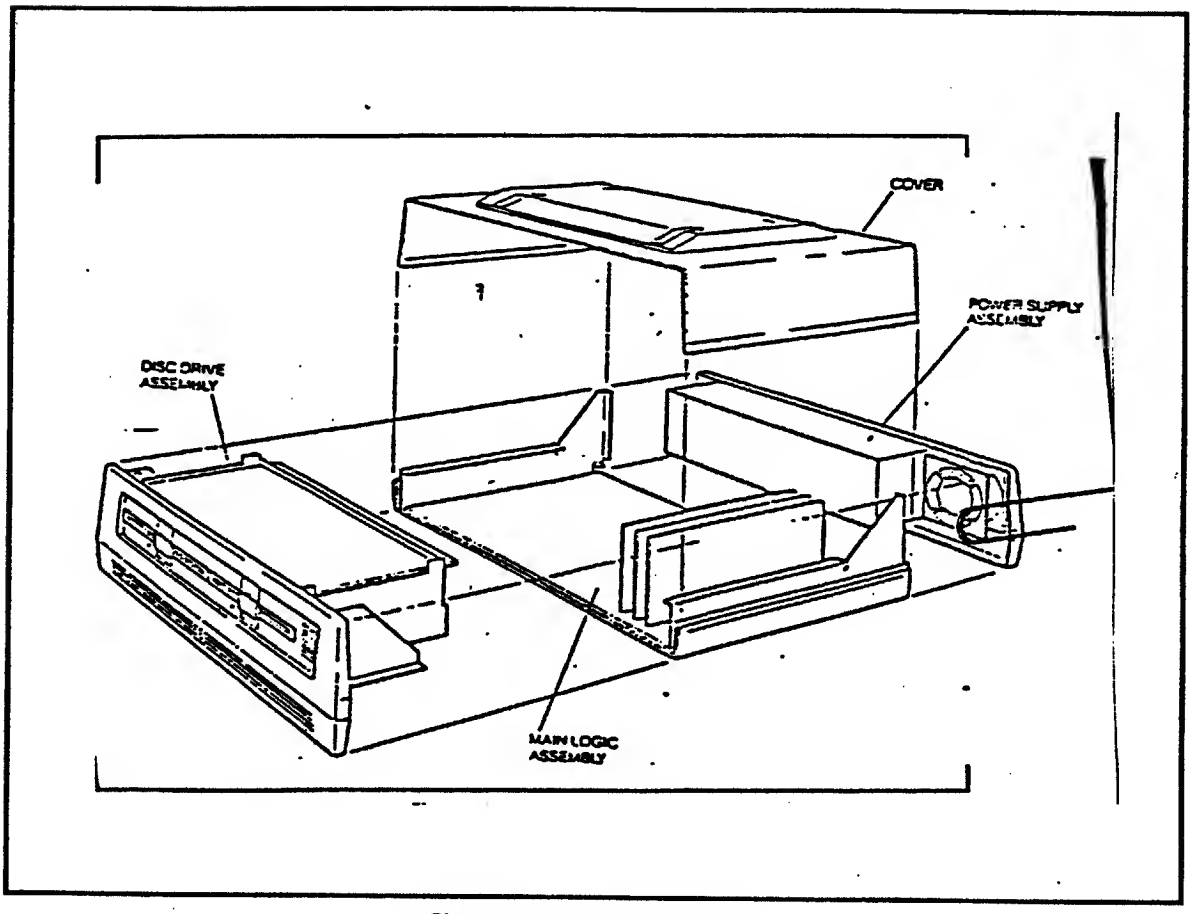

Figure 13-1. Processor Unit

13.2 Boot ROM

The VICTOR 9000 has up to 16K of boot ROM. When the 8088 is reset or powered on, the microprocessor goes to the highest memory area and begins to execute code in the boot ROM. The boot ROM performs basic initialization of all hardware in the machine. It then tries to read the boot software in the disk drives, which contains the operating system. The boot software is loaded into the processor's system random access memory (RAM). When this process is completed, the boot ROM jumps into the operating system and begins executing in the operating system.

POWER SUPPLY CIRCUIT

The ECR 2700 power supply produces 6 output voltages and 4 control signals that are distributed to the various circuits.

Supply Voltages

Voltage	Designation	Description
+18 Volts DC ±2V	VSO	Printer Voltage
-33 Volts DC ±3V (wrt vcc')	VPP	Display Voltage
+6.5 Volts DC ±1V	VCC'	Printer Driver Voltage
+5 Volts DC ±.5V	VCC	Logic Voltage
+5 Volts DC ±.5V	VRAM	Memory Voltage
5.5 Volts AC ±.5V	VF1 and VF2	Filament Voltage

ALL D.C. VOLTAGES WRT GND EXCEPT WHERE NOTED.

Signals

Designation	Description
Power Stop ($\overline{PS}$)	Informs CPU when power is failing
Reset ($\overline{RES}$)	Resets CPU at turn-on
Clock (CK)	Shift clock for reset circuit and reference frequency for buzzer
Chip Enable (CE2)	Used to enable or disable memory

CIRCUIT DESCRIPTION

VSO 18 Volt Supply

This circuit provides the voltages required for the printer motor, the solenoids that operate the print trigger magnet, stamp, paper feed and cash drawer open circuits.

115 VAC is applied to the primary of transformer T1, with 21 VAC being generated at the secondary winding. This voltage is full wave rectified by diode array, D1-D4 with the output at the cathodes of D1 and D4 approximately +30 volts DC unregulated. This 30 volt supply is filtered by capacitor C1 and supplied to the collector of transistor T1.

14. Sample Drivers and Device Handlers

To be supplied